Fundamentals of the Technique....

CD INCLUDED

MACHINE LEARNING WITH PYTHON

An Approach to Applied Machine Learning

- Introduction of Machine Learning
- Understanding Python
- Feature Engineering
- Data Visualization
- Basic and Advanced Regression Techniques
- Classification
- Un Supervised Learning
- Text Analysis
- Neural Networks and Deep Learning
- Recommendation System
- Time Series Analysis

ABHISHEK VIJAYVARGIA

BPB PUBLICATIONS

MACHINE LEARNING
WITH PYTHON

An Approach to Applied Machine Learning

by

Abhishek Vijayvargia

BPB PUBLICATIONS

FIRST EDITION 2018

ISBN: 978-93-8655-193-1

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

Distributors:

BPB PUBLICATIONS
20, Ansari Road, Darya Ganj
New Delhi-110002
Ph: 23254990/23254991

DECCAN AGENCIES
4-3-329, Bank Street,
Hyderabad-500195
Ph: 24756967/24756400

COMPUTER BOOK CENTRE
12, Shrungar Shopping Centre,
M.G.Road, BENGALURU–560001
Ph: 25587923/25584641

BPB BOOK CENTRE
376 Old Lajpat Rai Market,
Delhi-110006
Ph: 23861747

MICRO MEDIA
Shop No. 5, Mahendra Chambers, 150
DN Rd. Next to Capital Cinema, V.T.
(C.S.T.) Station, MUMBAI-400 001 Ph:
22078296/22078297

Published by Manish Jain for BPB Publications, 20, Ansari Road, Darya Ganj, New Delhi-110002 and Printed by Repro India Ltd., Mumbai

Special Thanks

To my father, S.N. Vijay
my mother, Meena Vijay
and my wife, Ankita Jain

This book is dedicated to my dearest parents and beloved wife who ever and ever supported me in all respects of life and career.

“Family is not an important thing. It’s everything.”

-Michael J. Fox

Preface

Machine Learning is the next big thing in computer science. From manufacturing to e-Commerce and shipping, it is used everywhere and provides solution which are based on the data. The idea behind Machine Learning is to make a knowledgeable Model which is as intelligent as human in taking a decision. Now with the increasing power of computation and memory, Machine Learning can provide even better solution than humans.

This book is organized in 12 chapters. Each chapter touches one of the primary area of machine learning.

First chapter of this book starts with basic Introduction of Machine Learning. Reader can get the idea of general Machine Learning Algorithms with their use to solve different types of real world problem. Chapter 2 contains basics of Python Language. Python is an open source language and a very good tool for applying Machine Learning. That's why we have chosen it to program Machine Learning Algorithms. Chapter 3 provides techniques to perform Feature Engineering. Finding correct features and modifying them is equally important in Machine Learning as Algorithms. Chapter 4 focused on Data VisualizaJon Techniques. With use of pre- built Python Library, user can visualize the data and present it to others.

Chapter 5 to 7 contains Supervised Learning Algorithms. Chapter 5 explains Basic Regression Techniques with example on a real-world data. Chapter 6 focuses on Advanced Regression Techniques with the solution of overfitCng problem. Chapter 7 provides details of classificaCon Algorithms. It contains both parameterized and non-parameterized techniques to solve the problem. Chapter 8 gives idea of Unsupervised Learning, mainly Clustering.

Chapter 9 to 12 provides knowledge on advanced concepts in Machine Learning. Chapter 9 covers Text Analysis. It contains an example to classify news in predefined category. Neural Networks and Deep Learning is discussed in chapter 10. This is highly used in unstructured, image and voice data. Chapter 11 provides methods of building a Recommendation System with examples in each category. Last chapter 12 discusses Time Series Data, methods to handle and make forecasting on it.

This book is helpful for all types of readers. Either you want to start in machine learning or want to learn the concepts more or pracIce with the code, it provides everything. We recommend users to learn the concept and practice it by using sample code to get the full of this book.

Although the content and codes are checked by other Data Scientists, there may be some shortcomings. Author welcomes your suggestions and criticism if any. Author will try his best to remove those errors in future editions of this book.

Acknowledgement

I would like to express my gratitude to the BPB Publication for bringing out the book in its present form.

A great thanks to my family and friends, without their support I wouldn't be able to complete this task.

"No matter how far I can go, my parents will remain always with me."

-Abhishek Vijayvargia

Table of Contents

Chapter 1

Introduction to Machine Learning

1.1 Introduction

We all are human. We learn from experiences. From the day we born, we start learning things. As we grow up, we start learning how to stand on feet and walk. We listen to the people around us and try to speak the same. We learn the meaning of different words. What to say when we are hungry or need something. We also start classifying things as good and bad. For example, when we go near to the fire first time, we feel the heat, and we come back. We learn not to go too close to the fire.

Now think about the working of a computer. It follows the instruction given by humans. It can process millions of instructions in a second and return the result. It can perform the task described by human but cannot take decision by itself.

Here comes the machine learning in action. What will happen if we will give computer ability to think like human? Isn't it Awesome? We can give every day's action in a format that computer can understand and do a math around it and build some model that will help it taking actions in future.

So, human learn from experience and computer follows instruction. Instead we can give experience directly to computer to learn and prepare itself for action. Now we define the experience in a structured format. So, we say computer learn from data (Experience) and this process is called **Machine Learning**.

Let's take an example of Banana shopping. Your mother instructed you to go to market and buy some good banana. She told you that bright yellow bananas are good. You went to a vendor and started picking banana as per your mother's advise. You took 20 bananas and came home. Now you noticed that some banana was not in good taste as others. In fact, five were bad ones. You took each banana one by one and started making assumptions. You understood that 12 bananas were big and 8 were small. All the 8 bananas were good but big ones were not same. So out of 12, five bananas tasted bad.

Next day you were ready with your knowledge. When you arrived in the market, you noticed some other vendor was selling bananas with discount. You went there and started picking small bananas. Now these bananas were different. This vendor had some green bananas. You took them. After reaching home, you again started with your banana testing skills and classify each as good or bad (depending on

taste). You found that big green banana was good but small green bananas did not taste as good as others. So, you learned new rule.

You started considering yourself as banana expert. One day you had to go to your cousins wedding in another city (far from your hometown). You saw bananas and went to test your skills. Now you surprised to see that all the bananas were very small and tasted very good (sweet like sugar). Here you learned that bananas from this part of the country were best.

Now you are an expert. You come home and ready for shopping. Your sister come home after a long time and she hates banana. She likes Guavas. Now what will you do? You again start your learning to find the best Guavas.

Now to make your computer do this task is Machine Learning. So, you give knowledge in form of data points. The property of data points is called **Features**. Here features are size of banana (small, medium, large), color, origin etc. And**Output** is taste (Good or Bad). So, you give this data to your machine learning program and it learns how to classify banana in good or bad category.

So, machine learning algorithms are kind of smart algorithms that gives you power of taking decision based on experience.

1.2 Machine Learning Process

Machine learning is not just a simple algorithm which you can put anywhere and start getting fantastic results. It is a process that start with defining the data and ends with the model with some defined level of accuracy. In this section, we learn about this process.

1. Define the problem

The machine learning process starts with defining a business problem. What is the need of machine learning? Does this task really need advanced predictive algorithm for solution?

Defining the problem is very important. It gives you direction to think about the solution more formally. It basically deals with two questions.

A. What is the problem?

This question covers the problem definitions and present it more formally. Consider a task where we want to find an image contains human or not.

Now to define it we will divide it in Task(T), Experience(E) and Performance(P)

- **Task (T):** Classify an image contains human or not.
- **Experience(E):** Images with the label contains human or not.
- **Performance(P):** Error rate. Out of all classified images, what is the percentage of wrong prediction. Lower error rate leads to higher accuracy.

B. Why does this problem need a solution?

This question focused more on business side. It covers the motivation and benefits for solving the problem.

For example, if you are a researcher and solving the problem to publish a paper and form a baseline for others, is may be your motivation.

Other needs is to identify, is there any human activity during night at bank's ATM when no security is present.

We also need to define the scenarios where we can use this solution. Is this a general solution or designed for specific task (i.e. detecting person in ATM sensors)? Also till when the solution is valid (is it lifetime or in a specific month/year)?

2. Collect the data

After defining the problem, data collection process starts. There are different ways to collect the data. If we want to associate review with ratings, we start by scraping the website. For analyzing twitter data and associate it with sentiment, we start by APIs provided by twitter and start collecting the data for a tag or which is associated with a company. Marketing researcher create different survey form and put it on the website to collect the data. In manufacturing industries sensors generate tera bytes of data per minutes. Websites generates logs on user activity. For big consumer companies like Amazon, Facebook this data is huge. Depending on problem, we also want to collect labels along with data. Suppose we want to build a classifier that classify news post to three groups, sports news, market news, political news. So, with each news that we have collected, we need one of the label associates with the article. This data can be used to build machine learning classifier.

So, right data is the key to solve any machine learning problem. More and better-quality data leads to generate better result even from basic algorithms.

3. Prepare the data

After data collection you need to focus on data preparation. Once you collect the data, you need to prepare it in the format used by machine learning algorithm. Algorithms are not doing any magic trick. You have to feed them with the right form of input to get the result. Depending on algorithm libraries, they expect different types of input formats.

Data preparation starts with data selection. Not every data gives actionable insights. Suppose we are analyzing logs on a server. It generates a lot of system related information after each user activity. That may not be useful if we are predicting the marketing response of a campaign. So, based on problem, we can decide to remove that data from further processing.

After identifying data on higher level basis, we need to transform or preprocess it to make it useful for machine learning algorithms. These are some of the process involves in preprocessing of the data.

- **Cleaning:** Data may have errors which needs to remove for processing. Suppose data have missing values for some attributes. Now some of the good algorithms cannot deal with missing values. So, we replace missing values

with some value (mean/median for numerical values and default for categorical values). Sometimes data contains sensitive information like email id and contact number of users. We need to remove it before sharing the data with the team.

- **Formatting:** Algorithm needs data in predefined format. Python based machine learning libraries expects data in the form of python list. Some real-time machine learning libraries use json format of data. Other, tools use csv for excel file. Depending on what tool or technique we are using, we need to format the data and put in the correct form.
- **Sampling:** Not all the data is useful. Specially for some algorithm which stores the data in the model, it is difficult to generate prediction in real time. We can remove the similar instances from data. If the data is labeled, we can remove the instances in the same proportion.
- **Decomposition:** Some features are more useful if decompose. Consider date attribute in a dataset. We can decompose the date in day, month and year. We can also create features like weekend or weekday, quarter of the year, leap year or with dates to make it more useful in predictions.
- **Scaling:** Different attributes follow different units and values. Suppose we are measuring height of a person in centimeters. For some data, it may be available in inches, so, first we need to transform that to centimeters. Also, higher/lower value of one attribute may affect the other attributes. For example, we have three features, such as person age, weight and annual income and we want to predict health insurance plan. If we use the data directly, model will depend highly on salary as the values are much higher as compare to other attributes. So, we need to scale each attribute to [0,1] or [-1,1].

This process is also known as **Feature Processing**. We work on feature select, preprocess and transform it in a form that is useful for Machine Learning algorithms.

4. Split data in training and testing

The goal of any machine learning algorithm is to predict well on unseen new data. We use training data to build the model. In training data, we move the algorithm in the direction which reduces training error. But we cannot consider accuracy on training data as the generalized accuracy. The reason is that the algorithm may memorize the instances and classify the points accordingly. So, to evaluate them, we need to divide them in training and testing. We train our algorithm on training data and calculate final accuracy by running them on testing data. Testing data is hidden to the algorithm at the time of training.

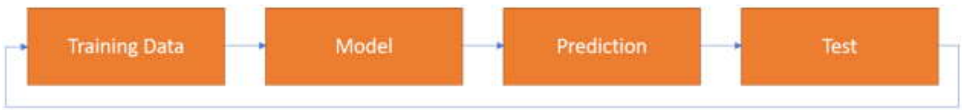
One general method is to use 60-80% data in training and rest of it as testing. So, the model which gives best result on test data is considered and accepted.

5. Algorithm selection

We start with our set of machine learning algorithm and apply on feature engineered training data. Algorithm selection depends on the problem definition. For example, if we are collecting data from emails and classifying them in spam or not spam, we need algorithms which takes input variable and gives an output (spam/not spam). These types of algorithms are known as **Classification** algorithm (Decision Tree, Naïve Bayes, Neural Networks etc.). If we want to predict any continuous variable (i.e. sales in an upcoming quarter), we use **Regression** algorithms (linear regression, kernel regression etc.). If our problem does not have any output or response associated and we can group them based on their properties, we use **Clustering** algorithms. So, there are bunch of algorithms in each category. We will see examples in next section.

6. Training the algorithm

After algorithm selection we start with training the model. Training is done on training dataset. Most of the algorithm start with random assignment of weights/ parameters and improve them in each iteration. In training algorithm, steps run several times on the training dataset to produce results. For example, in the case of linear regression, algorithm starts with randomly placing the separating line and keep improving itself (shifting the line) after each iteration.



7. Evaluation on Test data

After creating best algorithm on training data, we evaluate performance on test dataset. Test dataset is not available to algorithm during training. So, algorithm decisions are not biased by test dataset points.

8. Parameter Tuning

After selecting right algorithm for our problem, we start it and try to improve it for better performances. Each algorithm has different types of settings which we can configured and change the performance. This is called **Parameter tuning**. For example, for learning rate of algorithm, we can change the rate of learning and improve the performance. These parameters are called **Hyper Parameters**. Modifying these parameters is more like an art.

9. Start Using your model

After completing all the above steps, you are ready with the model which you trained and evaluated on your test dataset. Now you can use this model to start predicting the values for the new data points. For production environments, you can deploy the model to the server and use its prediction power by communicating

using APIs. Off course this model will not be always same. Whenever you get new data, you start by looking all the above steps to improve your performance.

So, in machine learning we start by our problem and finally finish with a prediction algorithm to solve the problem.

Let's start by some problem and try to see how machine learning algorithm can be used to solve this.

Suppose, you want to buy a house. You start looking at the houses available for sell in the market and check your budget accordingly. You may have multiple requirements that must be in the house that you will going to buy. But first start with something simple, size of the house.

You first check one house with 600 Square feet with the price of £220,000. That is nice but do not fulfill your requirement. Also, you want to move in with your family and this is very small house for everyone to live comfortably. So, you keep doing research and find another house of 1700 square feet with the price £730,000. That is very good, comfortable for your entire family but somewhat out from your budget right now. You can spend more money compare to small house price but not as high as the price of big house. You can get the actual price of the house only when you meet with the owner or agent and submit your detail. You don't want to do that for each house available for buy.

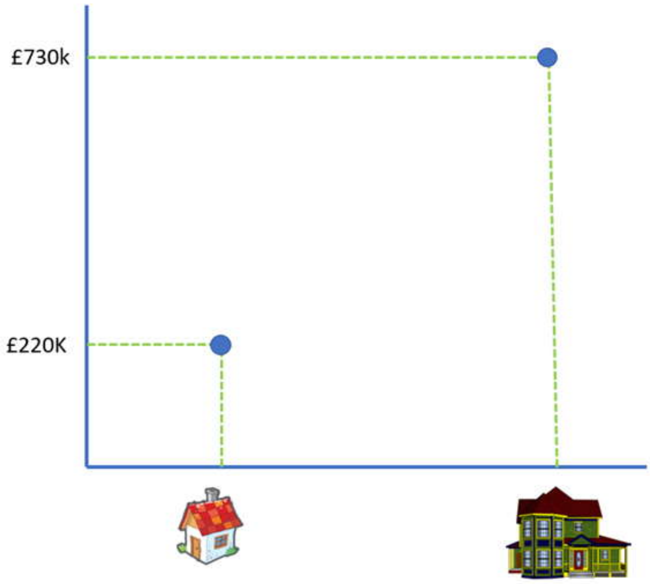


House Size = 600 Square Feet
Price = £220,000



House Size = 1700 Square Feet
Price = £730,000

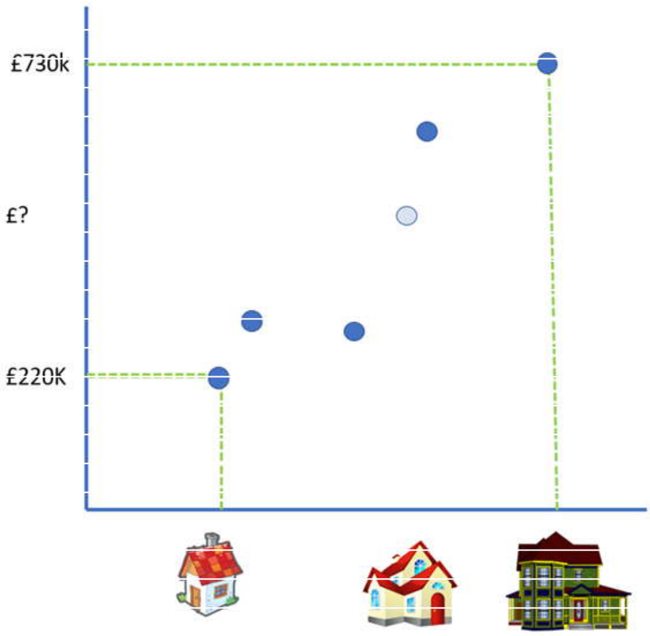
Let's analyze the two properties that you checked on the 2D plane.



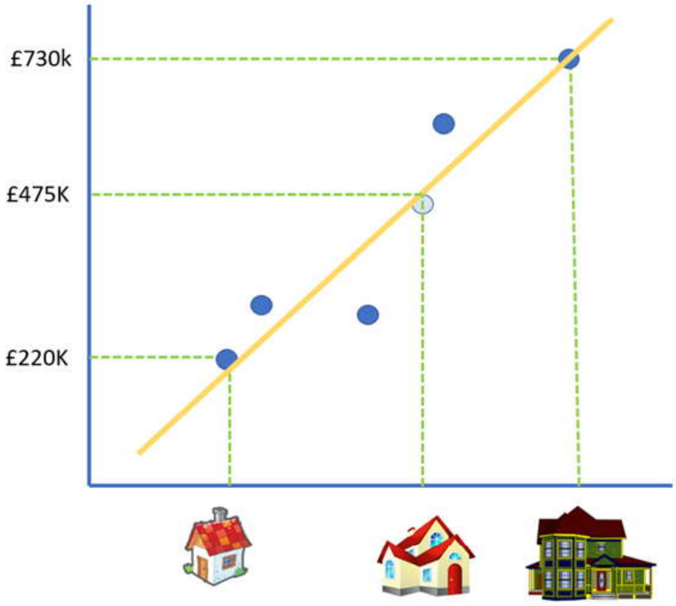
Now, you are roaming around the city and got one more house which has the size somewhat in the middle of these two houses.

This new house has the area approximate 1250 Square feet, but you don't know the price yet. So, you want to **Predict the price** and check that is it suitable to your budget and requirement?

Let's put it in the same 2D plane and try to predict the price for this house.

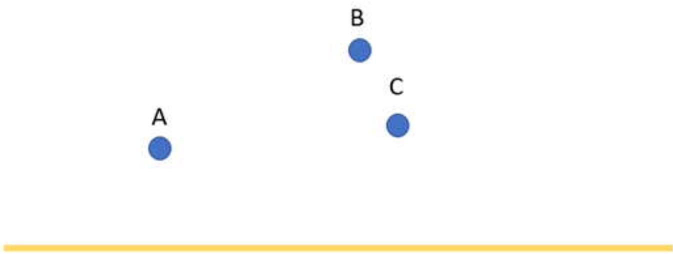


So, to do that we will fit the line that is made to satisfy the result which is known to us (i.e. maps price with house size known in the past.) We get the line like this.

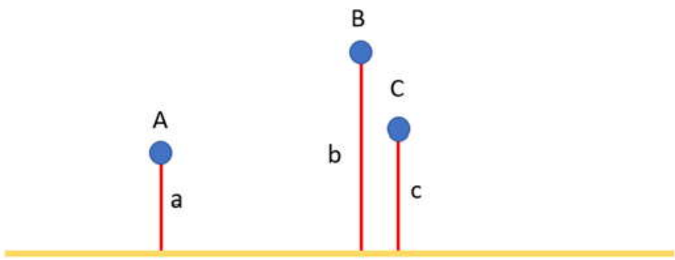


Using this line, you have predicted the price of the 1250 square feet house size is £475,000. So, we have formed a line to predict the price of the house by the size of the house. This technique is known as **Linear Regression**.

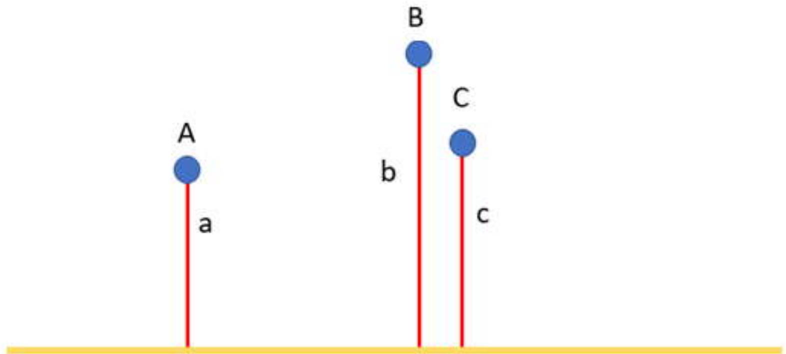
We can understand this technique to build the best line for available points. First, we start by randomly choosing a line.



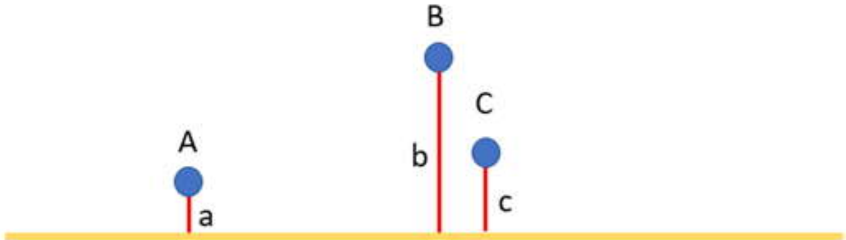
Now, calculate the distance from each point to the line.



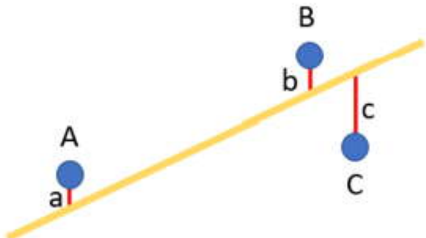
So, the overall distance is a+b+c. Now, let's move the line and calculate the distance again.



We have changed the line but the distance a+b+c increased. So clearly it is not helping. Let's move the line in other direction.

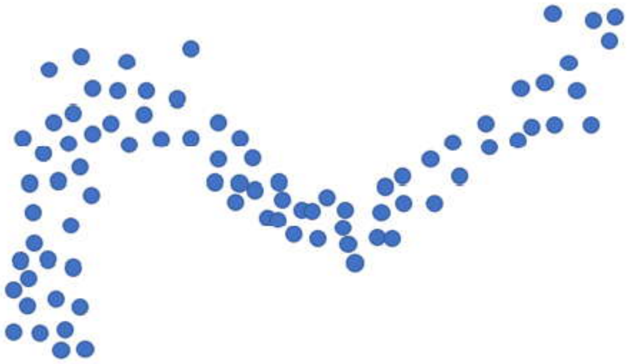


This is better than the first one. Now let's move this line and try to do the same step again and again. We will finally end up by using this assignment.

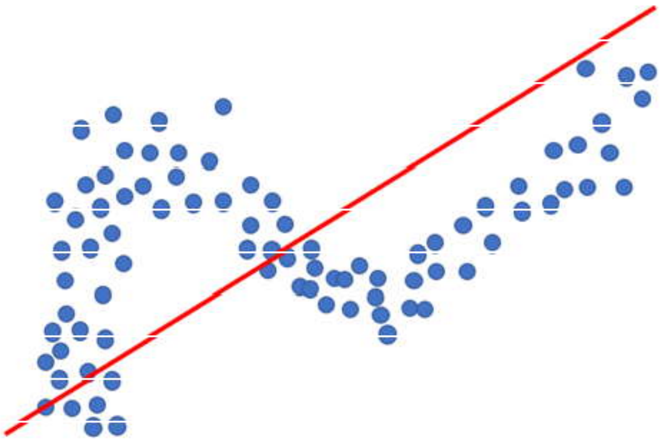


So, this is the most efficient line as per the given examples. Also, here we are taking non-negative distances from the line. The technique to find this line is known as gradient descent.

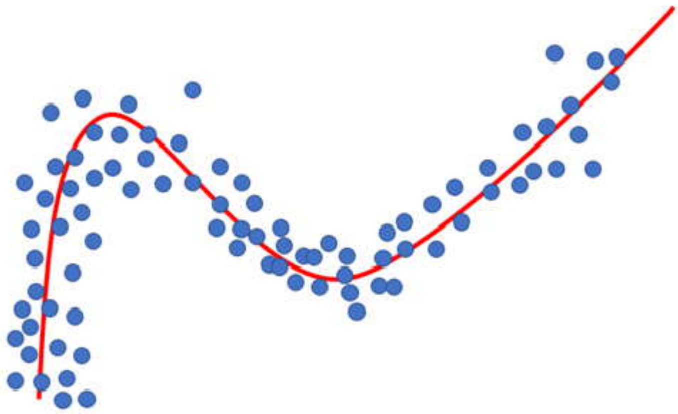
Sometimes fitting a line on all the points doesn't make much sense. Consider set of points as shown:



If we try to fit a line by using linear regression technique it looks like this:



Clearly this line is not great for making prediction. Instead, we use this curve to model the data.



This is known as **Polynomial Regression**. As the parameters are polynomial.

Let's move to another example. Consider a website which is selling shoes. It contains variety of shoes from different companies. You can order shoes from their online store. After every successful delivery of shoes, company sends an email to capture the feedback of customer. Customers write their reviews on the comment section and send it. Some reviews are positive, and some reviews are negative.

As the company sells thousands of shoes every day, it is needed to keep track of each review and take the action accordingly. For example, if customers are reporting for bad quality of a shoes, check with the manufacture about the issue. Or some shoes are getting very high response, better to put them on the first page of the website.

To solve this problem, you start with a set of customer comments. You classify each comment as negative or positive. Consider these examples

Positive

A1: Nice quality!! I love it

A2: Great product.

A3: Got one for my father. He loved it.

Negative

B1: Bad material. Not fit

B2: Hate this product. Packing was also pathetic.

B3: Never buy this product.

So, you start with these examples and start analyzing both positive and negative sets. You discovered that if the comment contains word 'love' it is more likely to be a positive comment. So, you make this rule and check over all the dataset. You find that 60% of the positive comment contains the word love. On the other hand, only 10% of the negative comments contains the word love.

Similarly, for other words

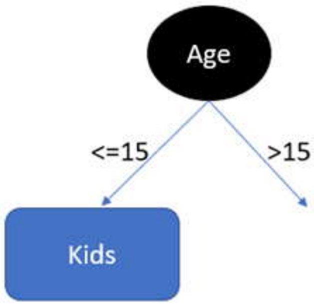
Word	Positive	Negative
Love	60%	10%
Great	45%	7%
Nice	36%	8%
Bad	4%	62%
Pathetic	2%	23%

So, for any future comment, we can multiple the probability and say it is positive or negative based on the words it contains. This is known as **Naïve Bayes Classifier**.

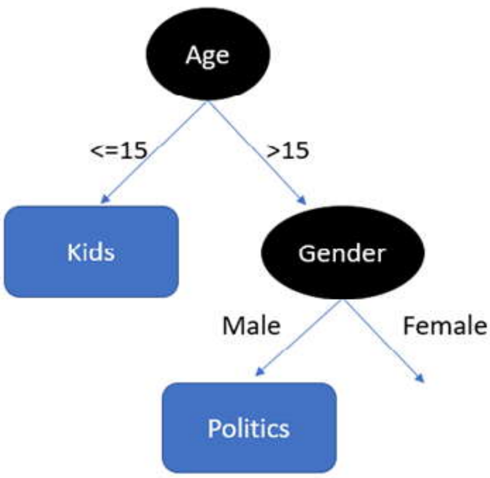
Let's move to another example of suggesting the magazine for different people. Let say we have recorded age, gender and location of users and the type of magazine they read. This is the data for that.

Age	Gender	Location	Magazine
21	Female	US	Sports
15	Male	US	Kids
37	Male	India	Politics
42	Female	UK	Business
32	Female	US	Sports
14	Female	India	Kids
53	Male	India	Politics

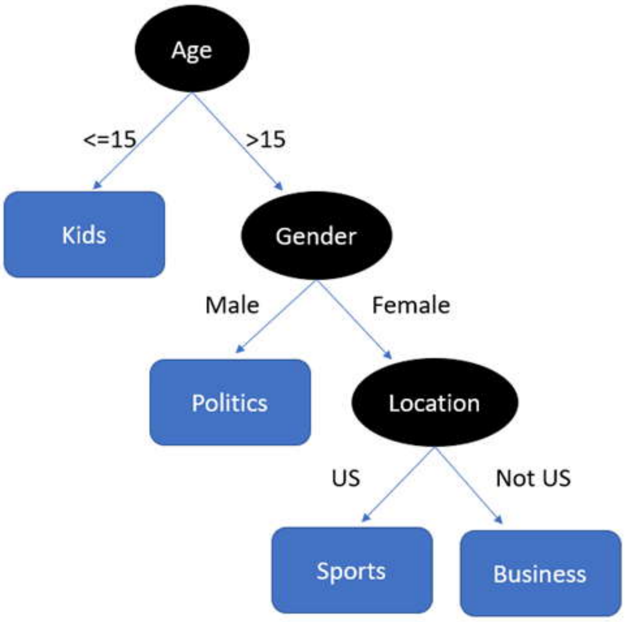
Now, by our observation from this data, we can see people which are less than 15 like to read Kids magazine. Let's make a node and its decision. Here a graph is presented. Each rounded node referred to decision node. Edges of the graph denotes the corresponding decisions. Each rectangle node represents decision taken by following that branch of the graph.



So, we can say each person with age less than or equal to 15 are likely to read Kids magazine. Now let's handle the other branch where people are more than 15 years. Now our second observation is male people, who like to read Political magazine. We create the same decision node and branches for it.

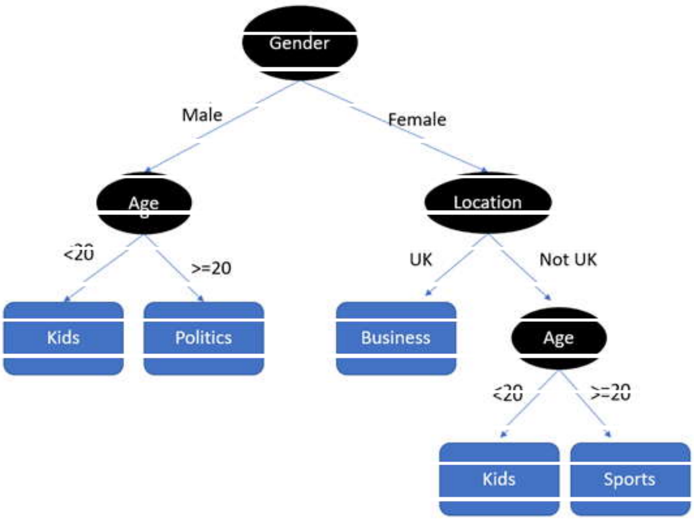


That is great. Now move ahead with the people whose age is greater than 15 years and Female. We still have one more information which we can utilize – Location. So, we can say people which are from US are like to read Sports magazine otherwise they like Business magazine. Let's form the node here.



We classified each data point correctly. So, this is called **Decision Tree**. There may exists multiple ways to create a decision tree. Those can also give correct prediction as per the available data.

Consider this tree

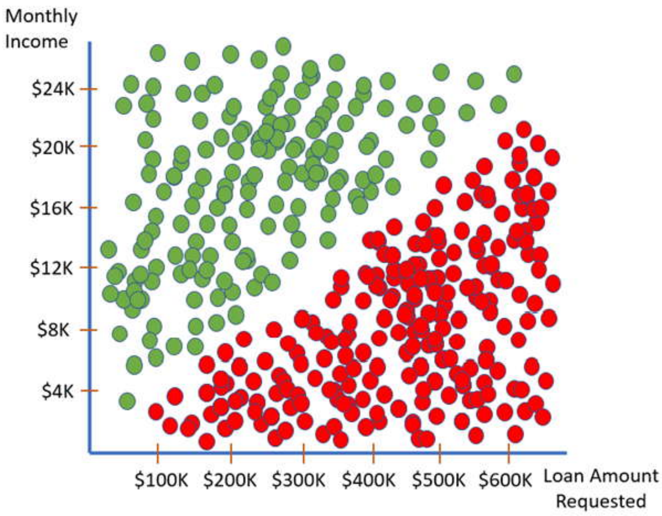


This also classify the data correctly.

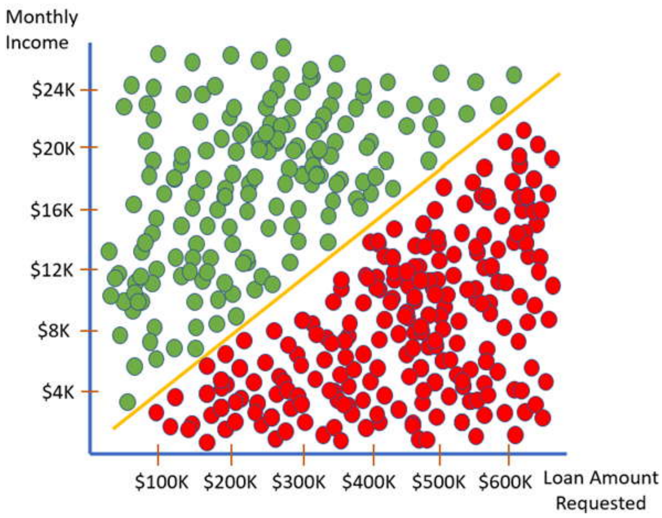
Let's move to another example. Several people apply for loan each year. Based on their income and loan amount bank decides whether to give loan or not. The objective is to give loan to those people who can repay the amount in the given timeline without any defaults.

For example, if a person has monthly income of $20K and applied for the loan of $100K, bank can grant the permission as they can see the source of income to repay the loan. On other way, if a person of income $3K per month applied for a loan of $600K, bank may reject his application early.

So, based on past defaulter history bank created a data which looks like this. Red points denote the application which got rejected by the bank. Green points denote the application which got approved by the bank.



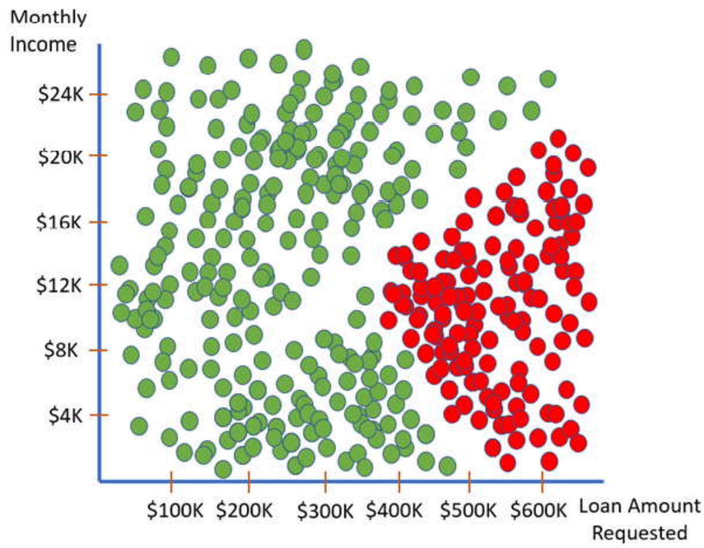
Now what happened to a person of monthly income as $10K and want a loan amount of $300K? Let's separate the data by using the line



From the above line we can say that the person with $20K monthly salary and $300K loan amount request is going to be accepted by bank.

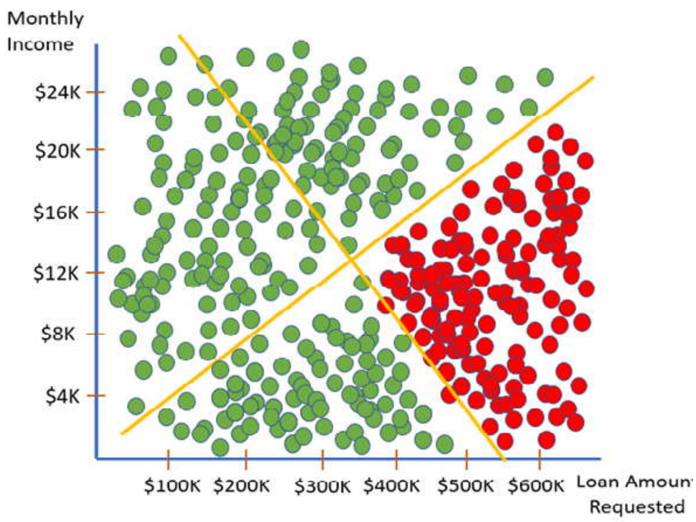
So, we have fitted a line to separate the data points. We used the same algorithm (gradient descent) which we use in the case of regression. Here, the target variable is categorical instead of continuous in the case of regression. This technique is known as **Logistic Regression**.

Now, some new manager comes in the bank and checked all the records, He thinks that the parameters by which bank is approving or rejecting the loan request is ridiculous. Some amount like $100K or $200K is not that risky and can be given to anyone with a valid request. So, he changed the rule and data looks like his.



Clearly, it is not possible to fit a single line to separate out red and green points.

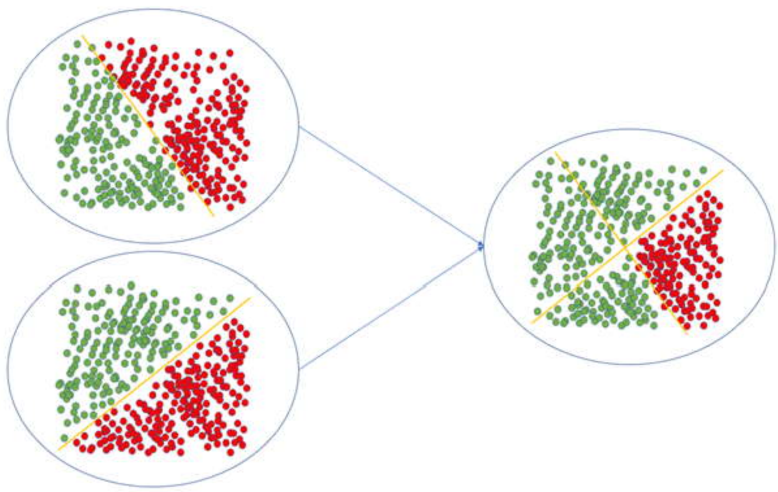
Ok, we can’t separate it by using a single line but what about two lines? Let’s see



This is Awesome. We have separated the Red points with Green points by using two lines instead of one. This technique is known as **Neural Networks**.

Neural network is based on the concept of neurons of our brain. In our brain neurons collects the information and pass it to other neurons. Based on the input from previous neurons, next neuron takes the call and decide the output. It also forwards its information to other neurons. Finally, by processing different neurons our brain takes the decision.

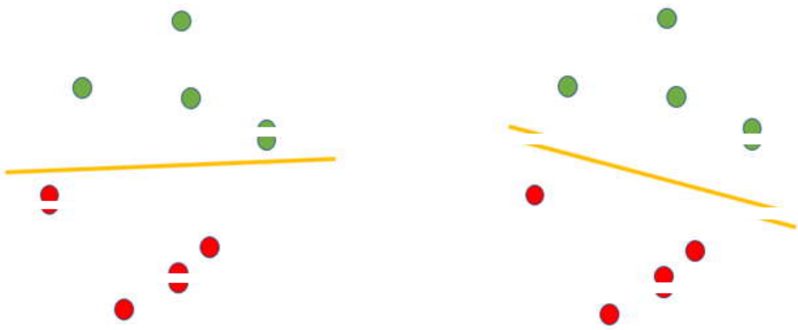
This concept can be understood by using the model as shown below, in which two neurons create model by using different assumption and forward their finding to another neuron. Output neuron takes the decision based on information collected.



Above are the representation of neural network by using different neurons.

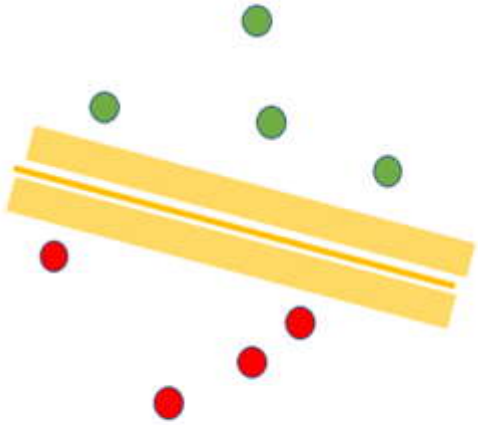
While handling data, we may have different choice for choosing a line that separate the data points.

Consider the two examples



Line 1 Line 2

Line 2 seems to be better in separating the data. It has more margins as compare to Line 1.

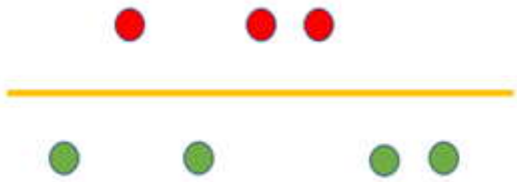


Finding the best line is better than any other separating line. This solution cannot be achieved by using gradient descent. We need linear optimization to find this line. This technique is known as large margin classifier or **Support Vector Machine (SVM).**

In real world, data is not clearly separable. It may come like this.



So, we cannot draw a line to separate red points with green points. But if we separate red points with green points by using a plane we can classify them by using a classifier. Let's create a new dimension and separate the points by using that plane.



Now our Red and Green points are separated by using a new dimension. This technique is known as **Kernel Trick**.

Real world data is much complex and has many dimensions. Kernel Trick with Support Vector Machine Classifier is used to solve a complex problem.

Now move to another problem. Consider the business of a grocery store A2A in a city. They provide call based delivery service. Whenever they get a call for any item they send their delivery boy with the address and the person deliver the product at the location. They managed to get their office in the center of the city from where they can serve more and more people on time.

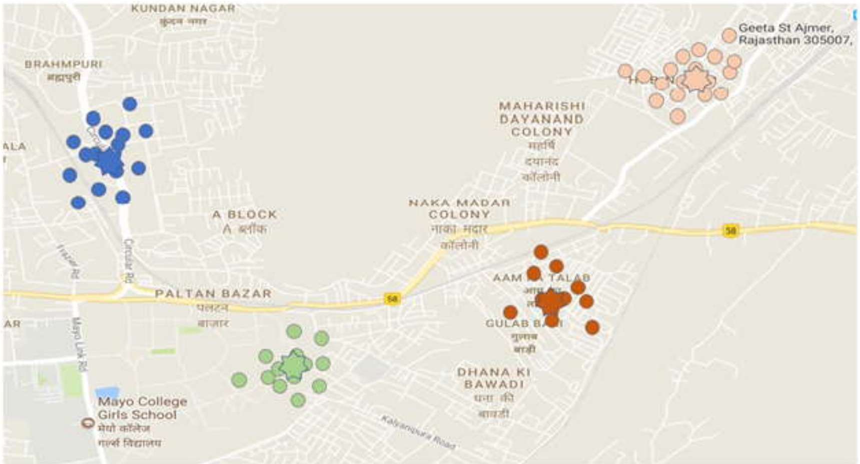


Here points represent service request came to A2A. Now they realize it would not make any sense to direct all request to one center. So, they decided to open four centers in the city which can take the request from different parts and solve accordingly.

So, to solve this problem, let's add the four centers randomly and put the request on them.



Clearly it is not the best assignment of delivery centers. In next steps, we move each point to the center of classified points. Again, classify all the points to nearest center and move store location to the center of classified step. After multiple iteration, we will get the assignment like this.



So, the assignment of each point is based on distance of points to center point. This technique is called **K-Means Clustering**.

We can use another method for clustering. Instead of first grouping them into clusters and reassigning, we can take all the points as independent clusters. Then, we group the two nearest points and form one clusters. We keep doing that till there is any large distance occur or we formed minimum number of clusters. This technique is called **Hierarchical Clustering**.

So, in this chapter, we have covered a lot of examples of machine learning in daily life. In this book, we will learn each one of them in detail and code in python. We will learn some advanced technique also.

Chapter 2

Understanding Python

To apply machine learning and data science a lot of tools are available. You can start solving machine learning problem by putting your data on those tools, performing some basic feature of engineering and start building predictive models. Most of the tools contain implementation of all popular machine learning techniques to solve any task.

Now consider a task where you have to solve some algorithmic problem. You need to perform some data filtering, apply sorting or searching and modification on the data. The best way to do that work is by programming language. Understanding programming language and its libraries gives you a power to solve any computational problem.

Consider the work of a mechanic in repairing a car. First, we go to mechanic and tell him our problem. He takes his tool kit and start searching right tools to complete the job. Then he starts applying those tools in order to complete the work. Not all the tools are necessary for this work. Some tools may need to be consider again and again to do the job. Also, some order should be followed to fulfill the requirement. For example, he cannot work on engine of the car without opening its front bonnet.

Real world problem solving also works in similar fashion. We first start by defining a problem. Than we start searching high level solution for the problem. After finalizing it, we move to find lower level solution and apply them in a particular order to solve problem. Similarly, Programming language works as a tool for us. We can implement our ideas/solution by using programming languages.

2.1 Why Python?

Python is very popular programming language among the data scientist. First, it is free and easy to learn. We don't have to pay much attention for small task. Second, it is object oriented, which is why it is supported by large organization developer community. Third, it's huge open source library support. Python has good number of library to achieve data science and machine learning solution faster. We don't need to write each algorithm, from scratch. Some of the libraries like pandas, scikit-learn, scipy, numpy, keras are very useful to implement your machine learning algorithm quickly.

Let's first understand the use of programming language to solve any real-world problem by an example.

Suppose you are hosting a party. You assign a number to each person coming in the party between 1 to 100 uniquely. Now when the party is over, you make an announcement.

"I will take out a random number between 1 to 200. If there exists any two people whose numbers sum is equal to this number, I will give a prize to both."

Now you know that you have distributed numbers to x people. How to ensure that you have to give prize to two people or not?

Let's try to solve this problem by using python. First, we start the problem by taking input data in a list.

```
numberList = list([43,23,1,67,54,2,34,56,23,65,12,9,87,4,33])
```

Now you choose a random number which we store in a variable.

```
sumSelector = raw_input()
```

Now to solve this problem, we have to sort the data first. In python list sorting is easy and can be done by sort function.

```
numberList = numberList.sort()
```

By using the above statement, we filled the variable with sorted number. So now we use two pointers. One from beginning and the other from end. We first check their sum. If it is less than the given value (means our current sum is less than the required sum) we move front pointer a step ahead and check. If it is greater than the given value (means our current sum is greater than the required sum) we move back pointer one step less and check. At any point if our current sum is equals to required sum, we can say that two people exists with the given sum and you have to give prize to both. Or if both pointer crosses their path and still the required sum is not achieved, you can claim that no two people exists and save yourself. The complete solution is written here

```
def isPrizeGiven(numberList,sumSelector):
    i=0
    j = len(numberList) -1
    if(i>=j):
        return False
    while(i<=j):
        currentSum = numberList[i]+numberList[j]
        print(i,j,currentSum)
        if(currentSum==sumOfTwo):
            return True
        if(currentSum>sumOfTwo):
            j=j-1
        else:
```

```
            i=i+1
    return False
```

To get the answer you have to pass your sorted list and sum to the function

```
if(isPrizeGiven(numberList,sumSelector):
    print("No Escape. Give Prize to Everyone")
else:
    print("You Saved Yourself!!")
```

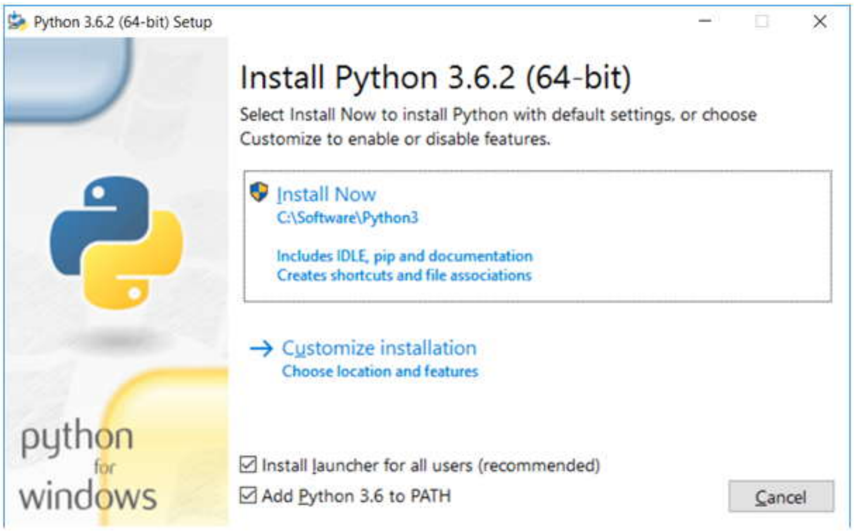
Don't worry if you are not able to understand the code. We will go through the entire process and understand basic principle of programming language.

2.2 Download and Install Python

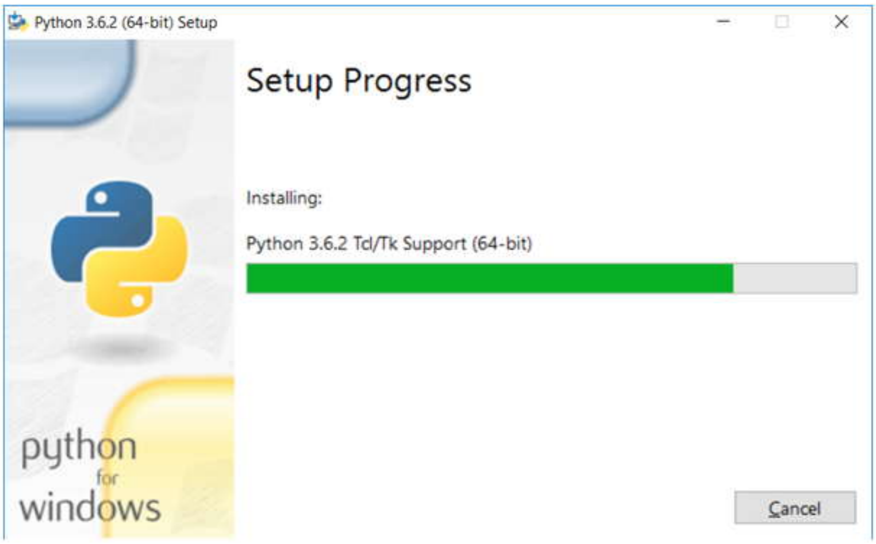
Python can be downloaded from its official website
https://www.python.org/downloads/
Depending on your operating system, you can download appropriate version of python.
When you first go to the website you can see two different versions of python (2.7.x and 3.x). Python supports multiple style of programming. User enjoys both Scripting and Object-Oriented way of programming in python. If you are a new programmer, we will recommend you to use Python 3.x. Although these versions are not much different with each other. As a note from same website Python 2.7, after releasing Python 3, no major version comes in Python 2.x. And Python 3 is growing, more libraries are added, so it is better to start with Python 3.

2.2.1 Install Python in Windows

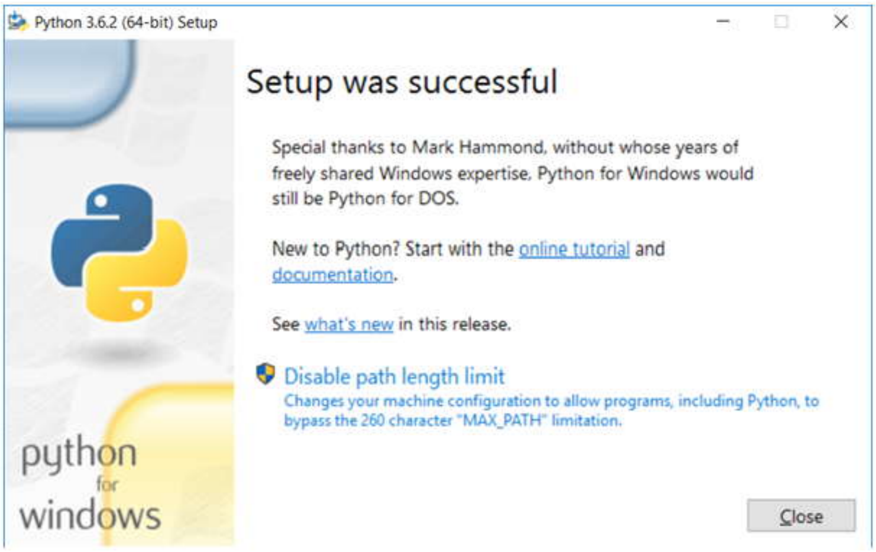
After downloading you can run setup file.



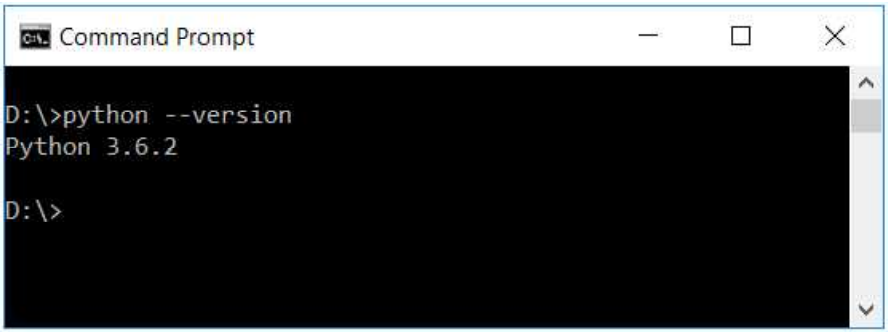
After this, click on Install now, installation begins. You can also choose customize installation. If you are using python first time, you can choose default setting.



After completing, the setup will process successfully



You can verify the installation by using following command



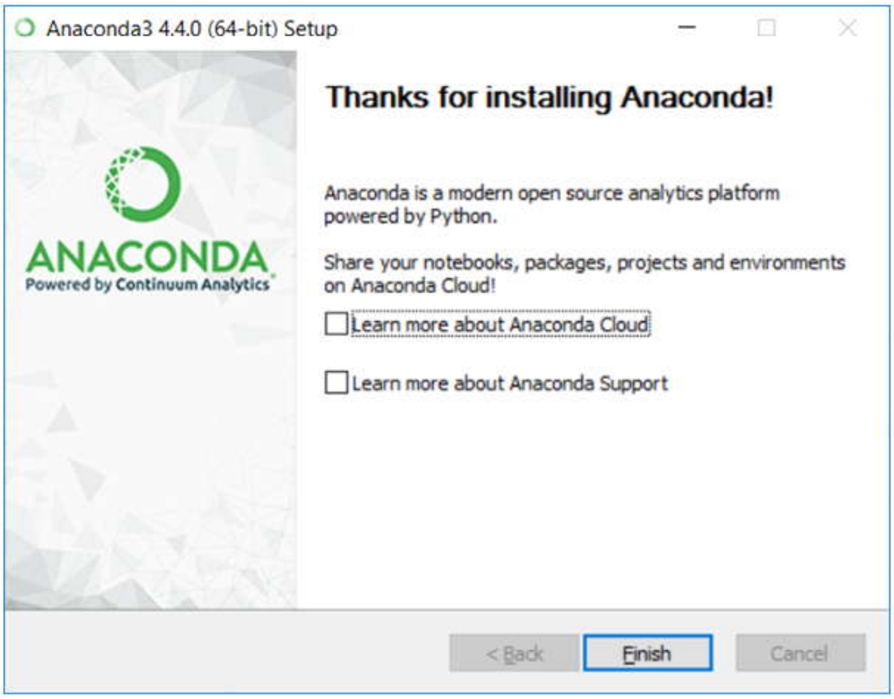
2.2.2 Anaconda

Anaconda is a useful package manager which contains all the useful data science library. The advantage is we need not to install all these libraries separately. Some of the important library which comes with Anaconda is pandas, scikit-learn, numpy, scipy etc. It also comes with Jupyter notebook and spyder for interactive development in python. Using jupyter notebook, we can write code, comment it properly and publish it. It also supports visualization packages support. We can see interactive graphs inline in the browser.

Anaconda can be downloaded from https://www.continuum.io/downloads.

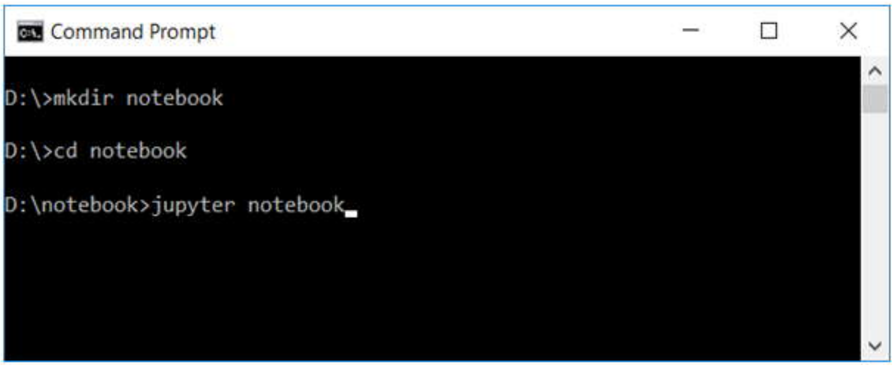
After installation, you need to run setup.



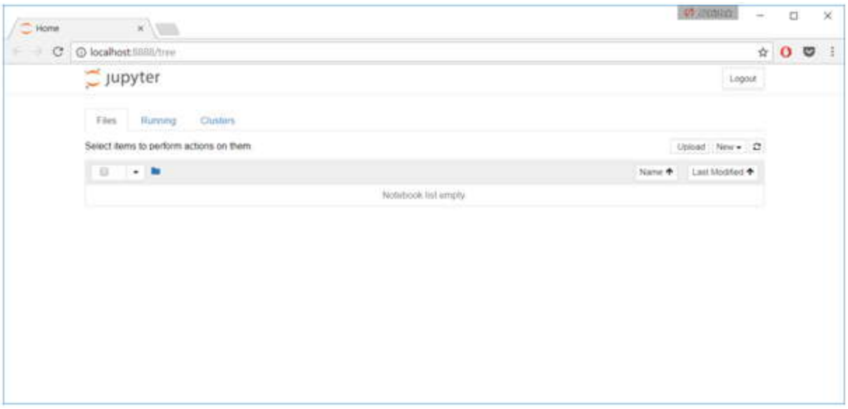


After installing anaconda, you need to add Anaconda folder and Anaconda/Scripts in the PATH.

jupyter notebook can be run by the following command.



After running the above command jupyter notebook will get open in your default browser



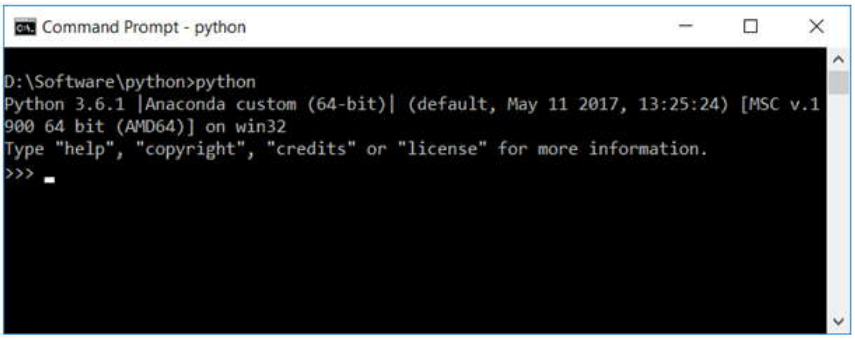
Now let's start learning python.

2.3 First Python Program

Let's code and run Hello World form python. Python programs can be written in different ways.

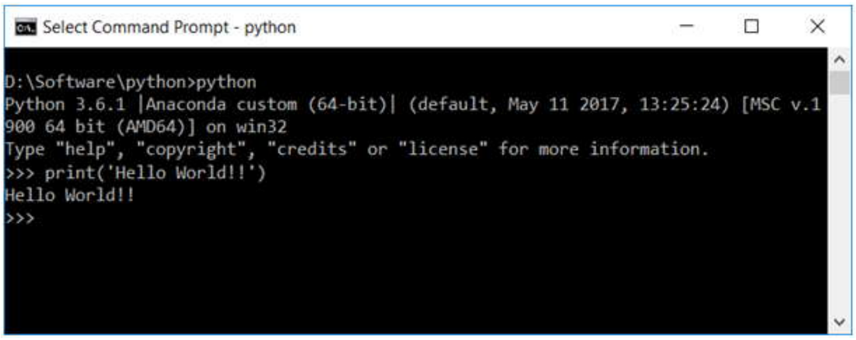
1. Start python from command prompt

Open command prompt, type python and hit enter.



Now python started.

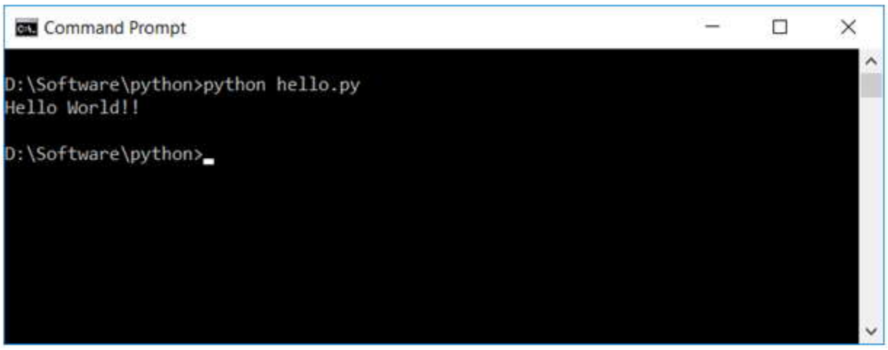
Let's print hello world, that can be done by using print statement in python.



You can exit from python by using exit() command.

2. Run a file

Python codes can be written in a file with .py extension. First create a file hello.py and add print('Hello World !!') in it. You can run this file by using python hello.py command.



3. Jupyter notebook

This is most interactive way to learn and test python codes. Open jupyter notebook by using command prompt and create a new notebook.



You can write the commands and run the cell by using Shift+Enter.

2.4 Python Basics

- **Constants:** Fixed values like numbers, strings are called constant because their value remain same.

```
print(82)
print(56.34)
print('I love Python')
```

- **Reserved Words:** These words are predefined in python and cannot be used as variable names. Some of the reserved words are: is, True, False, finally, for, try, except, import etc.
- **Variable:** It is a way to define some information in the memory and can be fetched later by using variable names.

```
a = 23
b= 'Python is Great!!'
```

This will first create a space in memory for number 23, store it in it and label that space with a. Similar things will be done for variable b.

If we reassign the variable, previous value will be removed, and new value will be stored at that place.

```
a = 43.34
```

Variable name can start with letters or underscore. It must consist of letters, numbers and underscores. Also, variable names are not case sensitive. ABC, Abc and abc are three different variables.

Correct variable names: abhi, str123, spammer_23, skynet

Incorrect variable names: 145etl, #Trad, re$5%

- **Assignment:** Assigning value to a variable by using assignment statements.

  It contains a value (or expression) on right hand and a variable on left hand. It assigns the right-hand side value to left hand side variable.

```
a = a + 5
```

In python we can assign multiple variables together

```
a = 9
b = 7
a ,b = 9,7
```

Both ways are same in python.

- Numeric Expression: It can be used in python to do the arithmetic.

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Remainder

```
a = 10
b = 5
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a**b)
print(a%b)
```

Expression are evaluated in terms of precedence. These are the order of evaluation:

- o Parenthesis
- o Exponentiation
- o Remainder, Multiply, Divide
- o Addition, Subtraction
- o Left to right

To understand precedence rule, take the example given below:

```
c = (5+2) * 4 - 7
print(c)
```

First parenthesis will be evaluated. Than the number is multiplied by 4 as multiply has higher precedence than subtraction. And finally, subtraction is performed and the result is stored in c.

- **Boolean and Comparison Operator:** Boolean variables have two possible values True or False. Let's take an example of Boolean variable.

```
is_male = True
is_senior_person = False
```

Both the above variables are Boolean variable. First is assigned with the value True and another is assigned to value False.

Boolean values are important in comparisons.

```
34>12
```

This will output True. We can put the result in a variable and use in conditional statements that we will cover in next section.

Following are the comparisons operators:

Operator	Symbol
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=
Equal to	==
Not equal to	!=

- **Strings:** In python strings can be created and assigned by using the single or double quotes.

```
valid_string_1 = "I am Valid!!"
valid_string_2 ='I am Valid too !!'
```

String can be concatenate using '+' operator.

```
first_string = "Abhishek"
second_string = "Vijayvargia"
concat1 = first_string+second_string
concat2 = first_string+" "+second_string
```

String size or length can be calculated by using len() function

```
len(concat2)
```

- **String Methods:** In python, string class has predefined methods which can be used to modify the strings. For example,

```
city_description = "Pune is a beautiful city."
print(city_description.title())
```

Output: Pune Is A Beautiful City.

Here, title is a method which capitalize first letter of each word in the sentence. We will learn more about these methods in Text processing chapter.

- **Type:** Python takes care of type of each variable. You can check it by using type() function.

```
type(c)
```

Python has many types. Let's print some of them

```
print(type(12))
print(type(12.45))
print(type('hello'))
print(type(True))
```

First is Integer, second is Float, third is String and forth is Boolean.

We can also convert some type to another. We can convert integer to float or string

```
a = 56
float(a)
str(a)
```

We can convert float to integer, but by doing this some information may loss.

```
b = 67.45
int(b)
```

We can also convert string to float or integers

```
str1 = '123'
str2 = 'Hello'
```

```
float(str1)
float(str2)
```

First statement executes successfully and convert 123 to float. Second statement fails because String 'Hello' cannot be converted to float.

In Python, user input can be taken by input() function. Input function returns a string. We can convert them to other type.

```
name = input('Please enter your name')
```

This will show 'Please enter your name' and waits for user to give input. That input is assigned to the name variable.

- **Comments:** We can add comments in python by using # at the beginning of line (or from where we start commenting). Text written under comments are ignored by python.
- **Errors in Python:** Python gives error when it fails to understand. Let's take an example of divide by zero.

```
34/0
```

This statement will output this in Jupyter notebook

```
-----------------------------------------------------------
ZeroDivisionError       Traceback (most recent call last)
<ipython-input-2-47ef12753bfd> in <module>()
----> 1 34/0
ZeroDivisionError: division by zero
```

Traceback means program stops because something went wrong. Whenever error occurred python stop processing next statements in the sequence and report error.

Last line shows what went wrong with python. It does not have value for divide by zero operation. One line above it with the arrow show in which line problem occurred.

- **Functions:** Using function, we can write code in blocks. Functions make code more readable and easy to reuse. Consider an example of function.

  Suppose we want to calculate the pay of person. Amount is the multiplication of number of hour worked and pay per hour.

```
amount = hours*pay_per_hour
```

Suppose we want to use this formula to multiple location in multiple files. We just copy this and paste to multiple locations.

Now one day your company decides to change the formula and give $500 bonus to each irrespective to their working hours.

```
amount = hours*pay_per_hour + 500
```

This is a big problem. You have to find each line of code and add 500 to it. Let's do it by using the function. Let's define the function.

```
def calculate_pay(hours,pay_per_hour):
    amount = hours*pay_per_hour
    return amount
```

Now we want to use the function, we just need to call this function with hours and pay_per_hour

```
calculate_pay(20,120)
```

Now by doing this. You can easily spot the block in your code. If you want to change the formula, you just need to go to the function definition and change it. That's it. It will be impactful throughout the code.

Now let's understand the different part of the function:

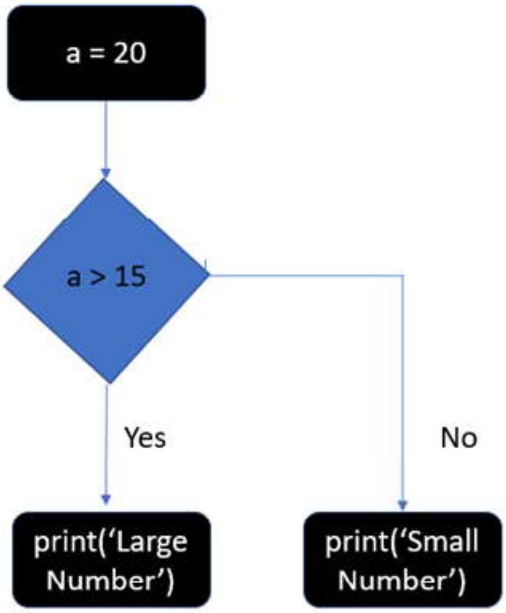
1. def is used to define a function. It is the way to tell the python about function declaration.
2. Following def is a function name. Rules for function name is same as of variable name. It must start with a letter or underscore and contains only letters, numbers and underscore. Here calculate_pay is the function name.
3. In the bracket parameters of the functions are given which are separated by comma. These are the values which are passed to the function. Function can use this information. Here we passed two argument hours and pay_per_hour. Function uses these two attributes to calculate the amount. It is not necessary to give the parameters always. We can put empty bracket (), if we don't want to pass any parameter to the function.

```
def print_welcome_message():
    print("Hello Stranger")
```

4. Function body consists code that executed on the function call. Each line is move by four spaces or a tab to declare as a part of function.
5. Function can have a return statement to return a value. We can use expression instead of variable in the return statement.

- **Conditional Statement:** In python, code run can be controlled by using conditional statements. Consider a scenario where you have to print 'Safe' or 'Unsafe' based on the configuration of a cart. Obviously, we can't print both. (We can but it is not logical to do that.) So, we need conditional statements. These statements are based on some conditions. Based on the condition true or false different actions can be taken.

Let's understand the concept by using the flow chart below:



First variable a is assigned to 20. Than a condition is checked whether a is greater than 15 or not. As the condition is true 'Large Number' is printed on the screen.

```
a = 20
if(a>15):
    print('Large Number')
else:
    print('Small Number')
```

If statement

Syntax of if statement is given by

```
if <condition>:
    do_something
```

1. Using if, we can tell python about conditional statement.
2. <condition> is the conditional expression. If it is True, code below the if statement is executed.
3. Colon is present after the condition.
4. Code in the if statement has four spaces or a tab at the beginning.

```
price = 200
if price<250:
    print('Buy !!')
```

More branches

In python we can create more branches by using else and elif (else if). Else code is followed by if block. Else block is executed when if condition is False. If the above condition is not executed check this.

if-else

```
price = 200
if price<150:
    print('Buy !!')
else:
    print('Sell !!')
```

if-elif-else

```
price = 200
if price<150:
    print('Buy !!')
elif price>=150 and price<250
    print('Hold !!')
else:
    print('Sell !!')
```

Here 'and' is used to check the both condition. If any condition is False, result of the expression is False. If both the conditions are True, result of the expression is True.

These are some of the Boolean operations

Operation	Result
a and b	if a and b both are True, expression is True. If a is False, result is False (without checking b)
a or b	if in a or b any one is True, expression is True. If a is False, then only b is checked.
not a	If a is True, False. If a is False, True.

Let's take example of each

```
a = True
b = False
if a and b:
    print('And')
if a or b:
    print('Or')
if not b:
    print('Not')
```

2.5 Data Structure and loops

Let's understand some basic data structure and looping in python.

List is a data type in python which contains comma which separate the values. Values are stored within squared brackets. Values need not to be of same type. It means a list can contain numbers, strings and another list in one single list.

```
my_first_list = [1,2,4,8,16]
```

List can be accessed by using index. Position starts from zero. It means first element of the list is at index 0 and last element of list is at the length-1 position.

```
print(my_first_list[1])
```

This will print 2.

We can also index the list from the end. Last element is at -1 position.

```
print(my_first_list[-1])
```

This will print 16.

Similar to strings, lists also has len() function.

List can be sliced like this

```
my_first_list[1:4]
```

Output: [2,4,8]

First number before the colon is start_index. Second number is end index. It is not necessary to give start_index or end_index both.

```
my_first_list[2:]
```

Output: [4,8,16]

```
my_first_list[:3]
```

Output: [4,8,16]

We can check element exists in the list by using in operator.

```
if 8 in my_first_list:
    print("Eight is Found.")
```

List Functions

max(list): This function is used to report maximum element from the list.

```
max(my_first_list)
```

min(list): This function is used to report minimum element from the list.

```
min(my_first_list)
```

sorted(list): This function returns sorted copy of list.

```
sorted(my_first_list)
```

We can sort the list in reverse order by setting True value in reverse parameter.

```
sorted(my_first_list,reverse=True)
```

list.append(element): Insert element in the list at the end.

```
my_first_list.append(32)
print(my_first_list)
```

list.insert(index,element): insert element in the list at specified index.

```
my_first_list.insert(2,74)
print(my_first_list)
```

list.clear(): deletes all elements from the list.

```
my_first_list.clear()
```

list.count(element): Number of times element appears in the list.

```
my_first_list.count(16)
```

list.remove(element): Remove element from list. Throw error if element is not available in the list

```
my_first_list.remove(74)
```

list.reverse(): Reverse the element of the list

```
my_first_list.reverse()
```

list.pop(): Removes last element from the list.

```
my_first_list.pop()
```

For loop

Loops are important to do the same task repeatedly. Consider an example of printing a number from 1 to 100. You don't want to write print() statement 100 times. The cleaner way to do in python is by using loops.

```
for i in range(1,100):
    print(i)
```

1. for is a keyword in python which is used to declare for loop.
2. After for is a condition for the iteration. It iterates i in the range of 1 to 100.
3. Colon after the iteration expression.

4. Body of the loop, indented by four spaces or a tab.

For loop on list:

```
for element in my_first_list:
    print(element+10)
```

While Loop

While loop is run for undefined number of times. It stops when condition is false.

```
i = 1
while i<5:
    print(i*i)
    i = i+1
```

1. While keyword is used to declare while loop.
2. Condition (i<5) that controls the loop. If condition is True, loop keep running.
3. Colon after the condition.
4. Body of the loop, indented by four spaces or a tab.

Use while loop wisely. Consider this example

```
i = 1
while i<5:
    print(i*i)
```

This is the infinite loops that prints 1 continuously. The reason is we are checking the condition but not updating the value of i.

Break

Break is used to come out of the loop. Generally, it is used with some conditional statement.

```
i = 1
while i<5:
    print(i*i)
    i = i+1
    if(i==3):
        break
```

This will print number 1 and 4. Whenever the value of i is 3, break statement is executed, and controls moves out of the loop.

Continue

Continue is used to skip the rest of the loop and transfer the controls to the top of the loop with next iteration.

```
i = 0
while i<5:
    i = i+1
    if(i==3):
        continue
    print(i*i)
```

This will print 1,4,16,25. When the value of i is 3, it skips remaining code and transfer the control for the next iteration.

Sets

Sets in python is a collection of unique elements in no order. We can consider it as unique element list with no order of elements defined.

```
my_duplicate_list = [1,2,3,1,2,3,5,6,3,2]
print(my_duplicate_list)
my_unique_list = set(my_duplicate_list)
print(my_unique_list)
```

Dictionaries

Dictionaries are key value pairs in python. At index the key values are stored.

```
dictionary = {'A':1,'B':2,'C':3}
```

Dictionary element can be accessed by using keys:

```
dictionary['A']
```

We can also change the value of dictionary element by using keys:

```
dictionary['A'] = 5
```

We can iterate over dictionary by using keys:

```
for key in dictionary:
    print(key,dictionary[key])
```

We can also define dictionary of dictionary:

```
complex_dict = {'first_dict': {'A':1,'B':2,'C':3},
'second_dict': {'E':1,'F':2,'G':3}}
```

Tuples

Tuples are ordered collection of items. Different between tuple and list is that the tuples are immutable, but list is not. Once assign we cannot insert or remove values in tuple.

```
dim = (45.23,67.34)
```

Tuples are useful when we have values which are used together.

Python Standard Library

Python library is organized in parts called modules. Before using any module, we need to import it.

```
import time
time.time()
```

Here, we used time module and get current timestamp.

One module contains multiple functions. Instead of importing all the function in that module, we can import specific function.

from module_name import function_name

```
from math import factorial
```

Here, instead of importing math module, program will import only factorial function.

We can use other names for module by using 'as' keyword

Import <module_name> as <new_name>

```
import numpy as np
```

Third Party Library

We can install other third-party library in python by using pip command on command prompt.

pip install <module_name>

pip install scikit-learn

Files

Files are used to store data on the disk. File can be anything like text, csv, image, audio, video etc.

File reading can be done by using read() function. First, we need to open file in read mode.

```
f = open('my_file.txt','r')
file_content = f.read()
```

After reading file can be closed by using close() function.

```
f.close()
```

File writing can be done by using write() function. We need to open it in write mode. Perform the write operation and close the file.

```
f = open('my_file_w.txt','w')
f.write('Writing in File')
f.close()
```

File can be read or write by using 'with' keyword

```
with open('my_file_w.txt','r') as f:
    print(f.read())
```

When the operation is performed, file is closed automatically.

Chapter 3

Feature Engineering

As the name suggest this process include engineering (creation, transformation etc,) of features from data. Now the first question is

3.1 What is Feature?

Consider an example of data stored in sql database tables. Table is made up of rows and columns. Table contains Integer data, String data, Date fields etc. Now consider the date column. We want to do some analysis, but this field is not directly useful. So, first we write a program (or script) to extract day on any particular date and create a separate column with that information. Now we have 7 days (Monday, Sunday etc.) stored in a new column. Again, we create a script to check: is a day is weekend or a weekday. We create another field is_weekend. It contains True if the day is weekend otherwise False. We can use this for our analysis or to build predictive model. This is **Feature**.

So, the form of attribute (or field) which is useful in data science process is called Feature. Feature may depend on the type of requirement. In the example above, we are more interested in getting the weekend information. Other analysis may need the day of week or month of year for analysis. In that case month of the year is the feature. So, this is completely application dependent.

Feature is used in:

1. In Computer Vision techniques image pixels, edges, corners etc.
2. In Speech recognition sound levels, noise levels etc.

3.2 Why Feature Engineering?

In Machine Learning to get success in any modelling technique

we need good features in the data. Features are very important to increase the predictive power of any model. When we try to solve real world problem, we may not always get the best features. As, Features may exist with a lot of problems like missing values, outlier, different type, error in data collection etc. Before training any machine learning model, we have to clean, transform and find right set of features.

We can take an example of cooking food. We have given a lot of ingredient and we have to find which is needed to solve our problem. Without selection of right ingredient, we cannot make perfect food. Same is feature engineering in data science process.

Data Scientist while solving any problem, spend more than half of their time in selecting the right features. Some of the benefits of spending time in feature engineering:

1. Make simple model to perform much better than complex model.
2. Reduce model selection time.They increase predictive power of simple models.
3. Reduce training time by simplifying the model.

It is all about decomposing the data in an intelligent fashion. We will discuss all the important aspects of feature engineering in this chapter.

3.3 Feature Extraction

Feature extraction is process of selecting new features from the existing set of features by doing some transformation in order to remove redundancy. In the original data, a lot of feature may exist. But it may possible that we can represent all the features with fewer new feature. This is called **feature extraction**. New features can represent the data very likely to the old features. The advantage of doing feature extraction is that the dimensions of the training data is reduced.

3.4 Feature Selection

Feature selection are the techniques to select subset of the features from the data. It is different from the feature extracted where we have created new features. In this we find most useful features from the data itself. Feature selection is important because:

1. Model creation is faster due to less features.
2. Easy to explain and interpret features.
3. Better Generalized model. Predictive model behaves nearly same on generalized data.

Feature selection is basically a search problem. We have to find ways to select features which can produce better results. Different techniques which are used in feature selection are:

1. **Filter Methods**: These methods are based on score of features by some statistical test. Each feature is evaluated with the outcome on statistical test (like pearson correlation, chi-squared test etc.) and a score is generated. Later features are ranked based on their score and lower score features are removed.
2. **Wrapper Methods:** These methods use machine learning algorithm to find the best feature. First, different subsets of features are created. Now with these set of features machine learning algorithm is trained on sample data and model performance is evaluated. The set of features which gives best performance are considered as selected features. These methods will take more time because they do actual training of algorithm with different set of features.

There are other techniques like embedded methods are used for feature selection. These techniques will select best feature during the training itself. Some of the examples are ridge and lasso regression which we will see in later chapters.

3.5 Feature Engineering Methods - General Guidelines

While dealing with different kind of features, we can follow some basic principles to achieve better results. These are some general guidelines which works well on almost all kind of data.

3.5.1 Handling Numerical Features

Data may contain a lot of numerical features. For example: cost of a product, temperature of the day, height of a person are all numerical features. A dataset is formed by mixing different kind of numerical attributes. Consider a problem of predicting a house. It contains attributes like house price, location, age of property, distance from nearest bus station, number of rooms as independent variable and rent as dependent variable (target variable).

If we take a raw data and apply machine learning algorithm, it is highly biased on the attributes like house price. Because the value of the price is domination and higher than other attributes like number of rooms. So, first we have to convert numerical feature to some standardize form to take it in the same range as other attributes. This method is applicable to all attributes. We call this process as **Normalization**.

Two popular techniques of doing normalization are

- **Min - Max Normalization:** This process brings the feature in the range of [0,1]. First it calculates minimum and maximum value of each numerical feature. After it apply the following transformation on each value of the feature

  $X_{new} = (X_i - \min(X))/(\max(X) - \mathrm{Min}(X))$

- **Z-Score Normalization:** Min Max normalization is not preferred when data contains outlier. In the presence of outlier, values are move to closer to zero as range of the data increases. Other techniques which is commonly used for doing normalization is z-score. Z-score follows the principle of statistics and move the data to mean zero and standard deviation as one.

  $Z = (X - \mathrm{mean}(X))/\mathrm{StdDev}(X)$

Both of the above techniques can be applied to normalize numerical features.

Sometimes, it is beneficial to convert numerical feature to categories or binning them together. This method also helps in reducing outlier effect and generate better prediction. For example, age of the property. This field can contain value like 1, 3, 5,15, 45 (In case of historical building). Now to understand the data properly we can bin this in 5 categories.

1. Less than a year (<= 1)
2. Between 1 to 5 years (> 1 AND <= 5)

3. Between 5 to 10 years (>5 AND <=10)
4. Between 10 to 20 years (<10 AND <=20)
5. More than 20 years (>20)

Above feature is just an example. Binning numerical attribute depends om different things like better generalization accuracy, correlation with target etc.

3.5.2 Handling Categorical Features

Categorical features are those features which contain value from a predefined set. For example, gender (Male and Female), colour (Red, Blue, Green), Noise Level (Very High, High, Good, Low, Very Low) etc. Categorical features bin the data in predefined categories.

Also, some problem may arise due to categorical variables. One problem is large number of categories. Consider an example of Indian cities. We have a data of 10000 rows and the city of India as one of the attribute (Pune, Ajmer, Jaipur, Mumbai, Kanpur etc.). Let say for 10000 row of data we have 500 different cities. Now, if we train this model and start predicting, our model is highly depending on the city. It overfits on training data to reduce the error and perform poor on generalized data. Also, if some city which is not a part of the current set comes, model fails to predict outcome for that.

To handle this situation, we can think of considering states instead of cities. So, we have to deal with only 29 states and 7 union territories. By reducing number of levels, we can create better generalized model.

Another problem is that some of the models only consider numerical attributes. Their cost function is based on the numerical value of feature. This problem can be handled in two ways:

- **Replacing each value with a number:** We can replace each category with a number. For example color (Red -1 , Green-2 , Blue-3). After making this transformation categorical attribute will contain only number and we can train our algorithm. But this technique will not always explain the data correctly. In above example we replaced Red with 1 and Blue with 3. As we know, all the three colors have equal weightage in original data. But here we are giving more weight to blue or in another way Green is near to Red as compare to Blue. Which is not correct.
- **Create dummy variable:** To deal with above problem, we follow dummy variable creation approach. In this method we create one new feature for each of the category. Each feature contains value of zero and one. If category is equal to the feature, it will get value 1 otherwise zero. In the case of color we will create three features (is_Red, is_Blue, is_Green).

Color	is_Red	is_Blue	is_Green
Red	1	0	0
Green	0	0	1
Blue	0	1	0

At one time, only one of the feature can take value 1. After creating new features, we can drop old feature (color in this case).

Also, we can do this task by creating number of categories - 1 features. In the above example we can skip creating is_Green variable. If the value of is_Red and is_Blue is zero, that means color is green. So, we are not losing any information but removing redundancy.

3.5.3 Handling time-based features

Time based features can be converted to categorical features based on the need. Consider a problem where time is given in the form hh:mm:ss dd:mm:yy (ex. - 22:45:12 18:12:2017) and we have to predict temperature at that time.

So, based on our knowledge, we know that the temperature of a day vary depending on time. So, we can extract time of the day and make a feature time_of_day which contains 4 values morning, afternoon, evening and night.

Also, we know that temperature vary in different part of the year. So, we can extract value of month of the year (1 to 12) and create a feature month and put the respective month (January, February etc.) in it. We can move one more level and create season feature (Spring, Summer, Fall, Winter) from the values of month.

Although for this kind of problem, one separate domain exists which is called **Time Series Analysis.** This we will cover in upcoming chapters. But if date is not our primary feature, we can do some transformation and make some good features out of it.

3.5.4 Handling text features

Text features are very common for real world application. Consider a problem of shopping cart website where you have to generate top products based on the user comments. So, for this you have to write a classifier which takes comments as input and classify them as positive, negative and neutral. And finally, you give a score (number of positive - number of negative) to the product based on all reviews and show high score products. Solution or predictive power of this model depends on the features from your text data. There are multiple approaches present to do that. We will also learn text classification in a separate chapter.

- **Creating a word count vector:** In this approach we create a set of all possible word which can come in the data. Second we create a matrix from data (rows are records and columns are count of word). We initialize this matrix with zero values. Now, we start processing each string and increase the count of words by 1 which comes in the string. Columns of this matrix acts as a feature vector in our problem

String	He	good	boy	nice	work	is
He is good	1	1	0	0	0	1
He is nice	1	0	0	1	0	1
Nice work	0	0	0	1	1	0

This is just a basic way to do things. We can apply a lot of other filters like remove stop words to improve the feature set.

- **TFIDF (Term Frequency, Inverse document frequency):** It is most commonly used method in text data handling. It is based on the fact that important words are those which comes frequently in related document but not comes in every document. So, applying this method each document/text get a score vector of words. Higher score of any word means word is important for that document.

3.5.5 Missing data

Missing data is very common problem in machine learning. Missing data means we don't get all feature value for some data points. We have to make some assumption or handle this case carefully. There are multiple ways to handle missing data.

- **Remove features which contains missing values:** One of the simplest way to analyse the data, is to check for all the features which contains missing value and remove them. We can train our model with remaining features. Whenever we get test data, we have to remove features from that also. This technique is useful when we have good set of features and only few columns contains missing value and their correlation with dependent variable is not high. But if most of the columns contains missing values, this technique is not preferred.
- **Remove rows contains missing values:** We can remove the rows from data which contains any data point missing. But if the test data also contains the missing values, our machine learning algorithm will fail to predict correct output. Also, if large number of rows contains missing values, this method will reduce training data.
- **Replace missing values with significant data:** This technique works well in most situations. Different types of data missing values can be replaced by different values. For numerical attributes, we can replace missing values by:
    - o Mean of the feature values.
    - o Median of the feature values.
    - o Zero.

For categorical attributes, we can create a new category 'default' for each missing data point and do the analysis.

3.5.6 Dimension reduction

Large dimensions of data are problem in model training. We can reduce dimension to a significant level by applying algorithms (ex- Principal Component Analysis, AutoEncoder etc.). These methods create new features that can explain full set of features efficiently.

3.6 Feature Engineering with Python

As we understand the basics of python, now we can proceed for learning advance python library to make our task easier. In this chapter we will focus on numpy and pandas. We will also see some basic techniques using **scikit-learn**. Scikit-learn contains model training algorithm which we will discuss in next chapters. After learning some basics of these library, we will do feature selection with the example of a dataset.

3.6.1 Pandas

Pandas is designed to do fast and flexible data access to relational data. Two primary data structures in pandas are Series and DataFrame. Series is one dimensional and DataFrame is two dimensional. For doing feature engineering, pandas has a lot of in built functions that can be applied directly on DataFrame and Series.

In this section we will learn general functions of pandas. First, we need to install pandas. We can do it with pip command.

```
pip install pandas
```

Pandas comes with default Anaconda package.

After installing, to use pandas we need to first import it. We can check for pandas version by using this command:

```
import pandas
print pandas.__version__
```

This command will print the version of pandas which are installed in your system.

3.6.1.1 Data Import

Reading csv file

```
csv_file = pandas.read_csv(fileName)
```

Above command will create a DataFrame and load data from file to it. We can check the type of csv_file object.

Above command has other options also. We can use them as per need.

- **sep:** separator of file. Comma by default. We can use any other separator which is used in our csv file structure.
- **names:** list of column names which is used as headers in dataframe.

- **nrows:** number of rows of the record to read.
- **na_values:** array of values which can be used as NA.
- **encoding:** encoding of data. Default is UTF-8.

```
type(csv_file)
```

Reading Excel file

```
exl_file = pandas.read_excel(fileName)
```

Reading json file

```
Json_file = pandas.read_json(fileName)
```

Reading saved object (python pickle)

```
pickle_frame = pandas.read_pickle(fileName)
```

Above method can be used to read DataFrame that is saved in memory for later use. We can create and do some manipulation on DataFrame and save it in the memory for later use. read_pickle is used for that.

Reading html

```
html_table = pandas.read_html(url)
```

This method read html tables from given url and create list of dataframe.

3.6.1.2 Data Export

Similar to data importing, we can also export the data in different format.

```
#Export dataframe to csv file
csv_file.to_csv(fileName)
#Export dataframe to excel file
exl_file.to_excel(fileName)

#Export dataframe to json file object
json_file.to_json(fileName)

#Export dataframe to pickle object
pickle_frame.to_pickle(fileName)

#Export dataframe to html table
html_table.to_html(fileName)
```

3.6.1.3 Data Selection

Pandas DataFrame is a collection of rows each having some predefined columns. Data selection includes technique to slice the DataFrame and extract information of need.

We will learn data selection by an example. First load 'product_data.csv' and create a DataFrame

```
product_df = pandas.read_csv('product_data.csv')
```

This file contains the following information:

	ProductID	ProductName	Cost	ShippingLocation	SellerName
0	1	LEDTV	32990	Delhi	ABC Pvt
1	2	Printer	5990	Delhi	ABC Pvt
2	3	Split AC	32050	Pune	XY Corp
3	4	Microwave	12670	Mumbai	PQ Corp

Series Selection (Column selection): This can be done by naming the name of the column with [ ].

```
product_df['ShippingLocation']
```

Above query will populate following result

	ShippingLocation
0	Delhi
1	Delhi
2	Pune
3	Mumbai

Multiple column selection: We need to specify list of columns.

```
listOfCols = ['ProductName','Cost']
product_df[listOfCols]

#Above line of code can be combined as
product_df[['ProductName','Cost']]
```

Output:

	ProductName	Cost
0	LEDTV	32990
1	Printer	5990
2	Split AC	32050
3	Microwave	12670

If we want to select data based on column

```
#For selecting rows with index = 2
index = 2
product_df.loc[index]
```

Output:

ProductID 3
ProductName Split AC
Cost 32050
ShippingLocation Pune
SellerName XY Corp

If we want to show all the rows whose index is less than some value

```
#For selecting rows with index less than 2
index = 2
product_df.loc[:index]
```

	ProductID	ProductName	Cost	ShippingLocation	SellerName
0	1	LEDTV	32990	Delhi	ABC Pvt
1	2	Printer	5990	Delhi	ABC Pvt
2	3	Split AC	32050	Pune	XY Corp

Sometimes index is not integers (it can be string, date etc.) If we want to access DataFrame by row we can use iloc.

```
#Select top 2 rows
row = 2
product_df.iloc[:row]
```

If we want to select rows from bottom

```
#Select bottom 2 rows
row = 2
product_df.iloc[-row:]
```

We can also put condition on selecting columns. We have to provide attribute for second dimension.

```
#Select all rows with 3 columns
product_df.iloc[:,1:4]
```

Output:

	ProductName	Cost	ShippingLocation
0	LEDTV	32990	Delhi
1	Printer	5990	Delhi
2	Split AC	32050	Pune
3	Microwave	12670	Mumbai

Conditional Selection: We can also select data by the condition:

```
product_df[product_df['Cost']>=30000]
```

Above will show all the product details having cost greater than 30000.

We can also use conditional operator to do selection.

AND Operation (Show the data which satisfy both conditions)

```
product_df[(product_df['Cost']>=30000) &
product_df['ShippingLocation']=='Delhi') ]
```

Above query will show all the product details which has cost greater than 30000 and shipping location is Delhi.

OR Operation (Show the data which satisfy any one condition)

```
product_df[(product_df['Cost']>=30000) |
product_df['ShippingLocation']=='Delhi') ]
```

Above query will show all the product details which has cost greater than 30000 or shipping location is Delhi.

Show the data which do not satisfy the condition

```
product_df[~(product_df['Cost']>=30000)]
```

Above query will show all the product details which has cost less than 30000.

3.6.1.4 Data Cleaning

Pandas DataFrame has good number of in-built function to clean the data. We can also perform sorting based on some column value.

Renaming variables

```
product_df= product_df.rename(columns={'Cost': 'price',
    'shippingLocation': 'destination'})
```

Insert new column with some value

```
product_df['NumberOfProduct'] = 1
```

Checking for null values

```
product_df.isnull()
```

Above will return same structure boolean DataFrame. If value is null, then the respective value is true otherwise false.

We can also check for any feature, which has null value or not instead of checking complete DataFrame

```
product_df['price'].isnull()
```

Dropping rows or columns with null values

```
#Drop all rows having any null value
product_df.dropna(axis=0)

#Drop all columns having any null value
product_df.dropna(axis=1)

#Drop all rows having less than k null values
product_df.dropna(axis=0,thresh=5)

#Drop all columns having all null values
product_df.dropna(axis=0,how='all')
```

Replace values

```
#Replace all occurrence of 0 with 1
product_df.replace(0,-1)
```

Filling NA values with some value

```
#Fill all NA values,
product_df.fillna(0)
```

Grouping data

We can do grouping of data same as we do in sql.

```
#Grouping on data by SellerName
product_df.groupby('SellerName')

#We can also group data based on multiple columns
regiment_preScore =
product_df.groupby(['SellerName','ShippingLocation'])
```

Apply Statistics on Grouped data

```
#Grouping on data by SellerName
product_df.groupby('SellerName')

#Now getting mean of Cost associated from each seller
product_df['price'].groupby(product_df['SellerName']).mean()

#Counting values
product_df['price'].groupby(product_df['SellerName']).count()

#Minimum Value of Cost of each seller
product_df['price'].groupby(product_df['SellerName']).min()

#Maximum value of cost of each seller
product_df['price'].groupby(product_df['SellerName']).max()
```

```
#Median of cost
product_df['price'].groupby(product_df['SellerName']).median()

#Std of Cost
product_df['Cost'].groupby(product_df['SellerName']).std()
```

Correlation of columns

```
#Correlation of numerical attributes
product_df.corr()
```

Get summary of data frame attributes

```
product_df.describe()
```

Sorting Rows

```
#Sort values by cost
product_df.sort_values(by='price')

#If we want to sort values in descending order
product_df.sort_values(by='price',ascending=False)

#We can also sort values by two or more features
product_df.sort_values(by=['SellerName','price'])
```

Checking type of features of dataframe

```
product_df.dtypes
```

Change type of feature of dataframe

```
#Convert Cost to String
product_df['price'].astype(str)
```

Above code will be useful if data has some invalid character in cost or sales field. After removing those data points, DataFrame column is still remains with the Object. So, we can change it to integer or float. That would be useful in getting statistics like mean, correlation or median of the data.

Combining Data Frames

In pandas we can add DataFrame in multiple ways

```
#To append one dataframe to other and store in third dataframe
product_df_complete = pandas.concat([product_df_1,
product_df_2])

#To append dataframe my columns (Similar to Join)
product_df_complete = pandas.concat([product_df,
product_new_info],axis=1)
```

```
#Merge dataframe
pandas.merge(product_df_1, product_df_2, on='ProductID')

#Outer Join
pandas.merge(product_df_1, product_df_2, on='ProductID'
how='outer')

#Inner Join
pandas.merge(product_df_1, product_df_2, on='ProductID'
how='inner')

#Left Join
pandas.merge(product_df_1, product_df_2, on='ProductID'
how='left')

#Right Join
pandas.merge(product_df_1, product_df_2, on='ProductID'
how='right')

#Append rows of dataframe (column should be identical)
product_df_2.append(product_df_2)
```

Now we will learn to solve common task related to feature engineering by using the above features and functions available in Scikit-Learn.

Apply and map function

We can apply functions on pandas dataframe.

```
#Declare lambda function
smallText =lambda x: x.lower()

#Apply on any column of dataframe
product_df['ShippingLocation'].apply(smallText)

#If we want to apply function element-wise
product_df.applymap(smallText)

#If we want to apply function on a series
product_df.map(smallText)
```

Difference between apply, map and applymap

apply: It can be used to apply function along with the axis

```
product_df['Cost'].apply(sum)
```

Above code will return sum of all the cost. If all the elements are numeric, we can apply it row wise also.

map: It takes the series and iterate over it. For example

```
product_df['Cost'].map(lambda x: x-50)
```

Above code reduces 50 from the cost of each item.

applymap: This can be used to apply function to each cell of the dataframe. Let's say our all cells are numeric and we want to get a square root dataframe.

```
Product_df_sqrt = product_df.applymap(sqrt)
```

Normalizing categorical features

1. **Min-Max Normalization:** This can be done using pre-processing functions available in scikit-learn

```
from sklearn import preprocessing

data = {'price': [492, 286, 487, 519, 541, 429]}
price_frame = pandas.DataFrame(data)

min_max_normalizer = preprocessing.MinMaxScaler()
scaled_data = min_max_normalizer.fit_transform(price_frame)
price_frame_normalized = pandas.DataFrame(scaled_data)
price_frame_normalized
```

In above code, we first create a dictionary that contains different price. Then we created a DataFrame from it.

One min max normalizer object is created, and data is given for doing the transformation. Here the maximum value transforms to 1 and minimum value transforms to zero.

Once min_max transformer, contains information, it can be used for transforming test data.

```
scaled_data = min_max_normalizer.transform(test_frame)
```

2. **Z-Score Normalization:** This process is based on mean and variance of data. After normalizing, mean of the data is 0 and variance is 1.

```
from sklearn import preprocessing

data = {'price': [492, 286, 487, 519, 541, 429]}
price_frame = pandas.DataFrame(data)

min_max_normalizer = preprocessing.scale(price_frame)
price_frame_normalized =
pandas.DataFrame(min_max_normalizer,columns=['price'])
price_frame_normalized
```

Binning Numerical Features

Using sklearn preprocessing, we can bin numerical features to fix categories. For example, we have transformed age values to four categories.

```
from sklearn import preprocessing

data = {'age': [28,27,31,31,15,41,61,26,85,9,88,51,26,52,19]}
age_frame = pandas.DataFrame(data)

bins = [0, 20, 40, 60, 100]
group_names = ['Less Than 20 years', '20 to 40 years', '40 to
60 years', '60+ years']

age_frame['categories'] = pandas.cut(age_frame['age'], bins,
labels=group_names)
age_frame
```

Converting Categorical variable to Numerical

We can convert categorical variable to numerical feature by assigning each category a number. This can be done by labelEncoder in sklearn preprocessing.

```
from sklearn import preprocessing

encoder = preprocessing.LabelEncoder()
encoder.fit_transform(["Delhi", "Pune", "Mumbai", "Delhi"])
```

Above code will assign one integer to each unique category starting from zero.

We can also perform reverse transformation and get the category back from labelEncoder.

```
encoder.inverse_transform([2,1,1,0])
```

Creating dummy variable from categorical data

This process will create new column for each categorical data. After creating we can combine this DataFrame with original DataFrame.

```
dummy_frame = pandas.get_dummies(age_frame.categories)
dummy_frame
```

Extracting information from date features:

Date features are important. They contain useful information which can be used either directly or by some modification. For example, predicting crowd at any good restaurant is depends on the day. If it is weekend, intensity of crowd is high. Also, for daytime in weekday, most of the seats are empty. So, to extract these kind of information, we have to analyse dates.

Let's create the data first and load it in a DataFrame

```
import pandas
startDate = '2017-07-06'
endDate = '2017-12-18'
Dates = pandas.DataFrame(pandas.date_range(startDate,
endDate),columns=['Dates'])
```

Now extract date month and year and create new features.

```
import numpy
Dates['day'] = Dates['Dates'].map(lambda x:x.day)
Dates['month'] = Dates['Dates'].map(lambda x:x.month)
Dates['da_of_week'] = Dates['Dates'].map(lambda
x:x.weekday_name)
Dates['is_weekend'] = Dates['Dates'].map(lambda
x:numpy.is_busday(x, weekmask='1111100' ))
```

In the above example we used is_busday function of numpy library. This function is useful to check that date is a business day or not. weekmask is a string of ones and zeros. One for each business day starting from Monday and zero for other. We can also give list of holidays in this function.

Similarly, we can also do other transformation to get useful features as is_winter, is_summer, is_night, is_morning etc. depending on requirement.

Creating word count vector

Text data can be handled by creating one feature for each word. For each text instance, feature contains count of that word in the text.

```
from sklearn.feature_extraction.text import CountVectorizer
comments=['nice product',
    'bad condition',
    'shiiping was bad',
    'great delivery']
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(comments)
word_frame = pandas.DataFrame(X_train_counts.toarray())

#Get all column name and integer index mapping
word_dict = dict((v, k) for k, v in
count_vect.vocabulary_.items())

#Now replace integer column names with words
word_frame = word_frame.rename(columns=word_dict)
word_frame
```

Creating TFIDF vector

Term frequency Inverse document frequency vector can also be created by using sklearn. It takes count vector and create TFIDF vector.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_transformer = TfidfVectorizer()
#tfidf_transformer.fit(X_train_counts)
X_train_tfidf = tfidf_transformer.fit_transform(comments)
```

```
word_frame = pandas.DataFrame(X_train_tfidf.toarray())
#Get all column name and integer index mapping

word_dict = dict((v, k) for k, v in
tfidf_transformer.vocabulary_.items())

#Now replace integer column names with words
word_frame = word_frame.rename(columns=word_dict)

word_frame
```

Dimensionality Reduction

Dimensions can be reduced by multiple techniques. We will discuss these in later sections. One simple example illustrating PCA is shown below:

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit_transform(word_frame)
```

Chapter 4

Data Visualization

Data visualization is graphical representation of data. It is easier to understand the result by charts and graphs rather than reading tables or thousands of lines of raw data. Our brain can understand visual things better and it helps to take the decisions.

For example, Consider the way a human and computer take the decisions. Computer can process information much faster with large amount of data. Suppose we have written a program to read millions of lines of data and take a decision. For humans, it is not that easy. They need time to analyze the data and make decision. To deal with this situation, we present this data visually. For example, using bar chart to see the impact of each decision in past or pie chart showing past decision and their success.

At the end, data scientists have to show their results to the audience, who do not have knowledge of machine learning algorithms, hypothesis or data engineering. Data Visualization helps in that way to convey the decision makers to take the decision. Data scientist used this as a tool to build a story around the data.

In this chapter, we will learn how to generate graphs and charts in python. We will use functional and object-oriented approach to visualize the data.

Some of the popular packages in python to visualize the data are:

- matplotlib
- seaborn
- ggplot
- geoplotlib
- bokeh
- plotly

We will use matplotlib in this chapter. Also, we will learn to code other useful libraries.

4.1 Line Chart

It is a chart in which series of data points are connected by straight lines. It provides simple behavior of a parameter over the other. This is most commonly used to show the trends over time. We can use line chart to compare related features.

Let's start by generating our first chart in jupyter notebook.

First import the library needed for this:

```
import matplotlib.pyplot as plt
```

By default, matplotlib open new browser tab to show the result. If we want to see the result in line with the current notebook, we can use this command:

```
%matplotlib inline
```

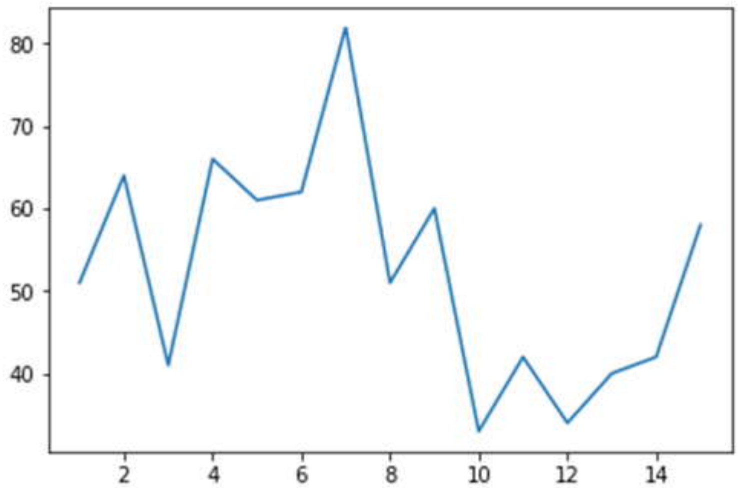
We need data along with two axes. On x-axis take number from 1 to 20. On y-axis take random data with mean = 50 and standard deviation = 10

```
#On one axis number from 1 to 15
a = range(1,16)

#On other axis generate random integers with mean and sigma
mean = 50
sigma = 10
b = numpy.random.normal(mean, sigma, 15).astype(int)
```

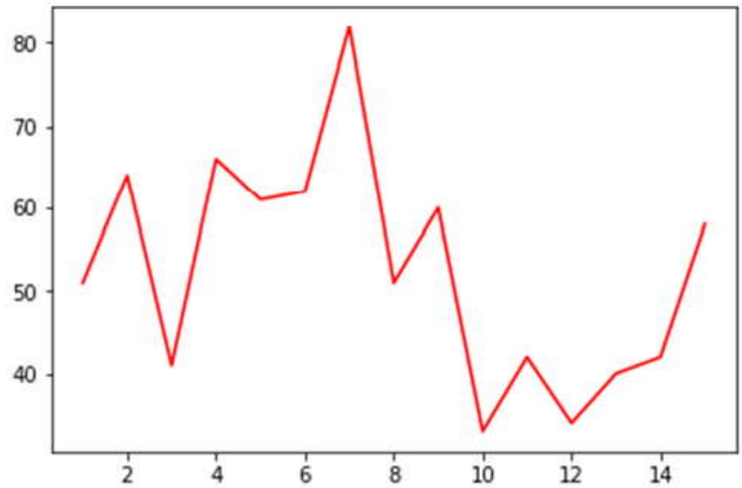
Now, just apply plot command and the following line chart will appear.

```
plt.plot(a,b)
```



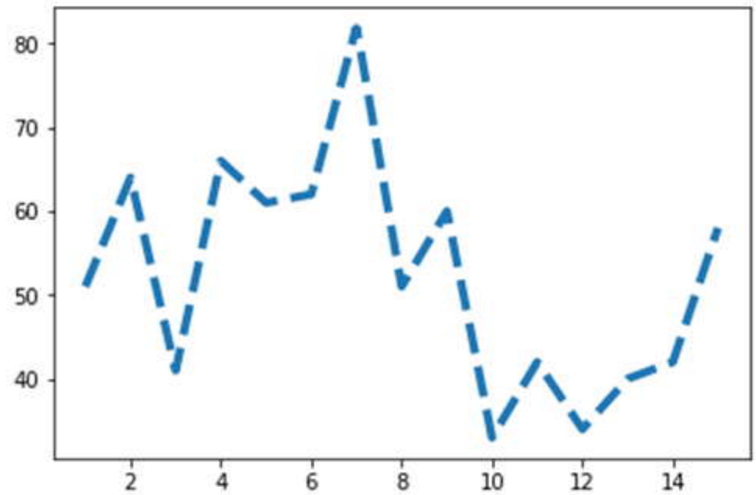
We can also change the color of lines using following code:

```
#Define color
plt.plot(a,b,color='Red')
```



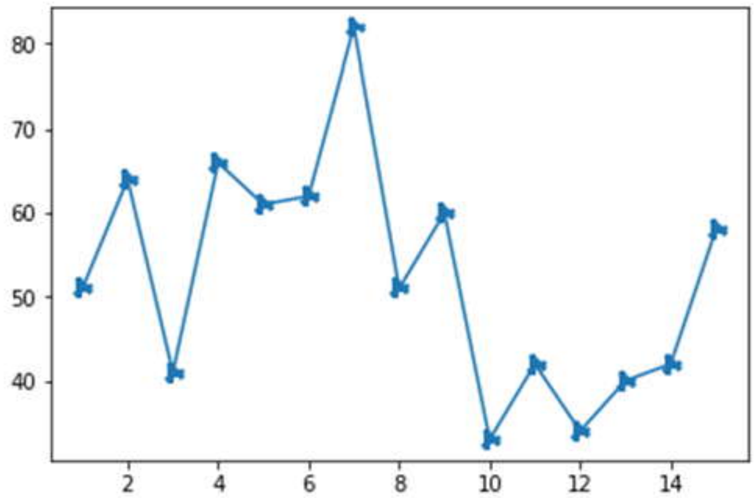
We can change the type of line and its width by ls and lw variables:

```
#Change line style and width
plt.plot(a,b,ls='--',lw=4)
```



Markers can be added on each data points by:

```
#Defining markers and marker width
plt.plot(a,b,marker='3',mew=10)
```

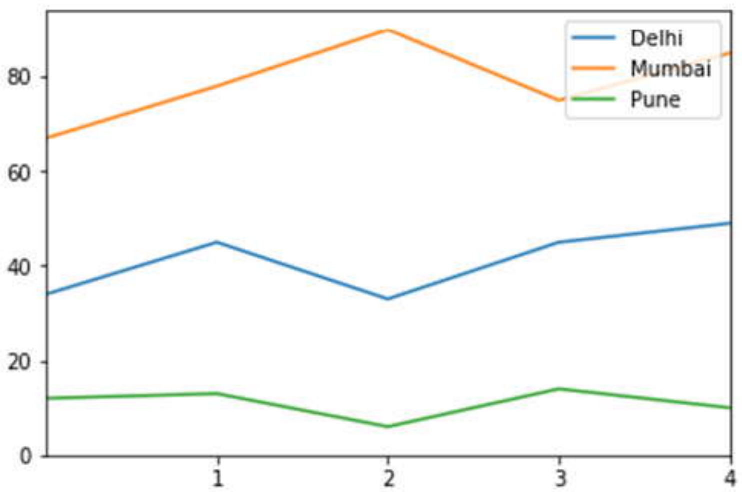


We can also plot line chart from pandas DataFrame:

```
sales_in_Delhi = [34,45,33,45,49]
sales_in_Pune = [12,13,6,14,10]
sales_in_Mumbai = [67,78,90,75,85]
sales =
pandas.DataFrame({'Delhi':sales_in_Delhi,'Pune':sales_in_Pune,'Mumbai':
sales_in_Mumbai})
```

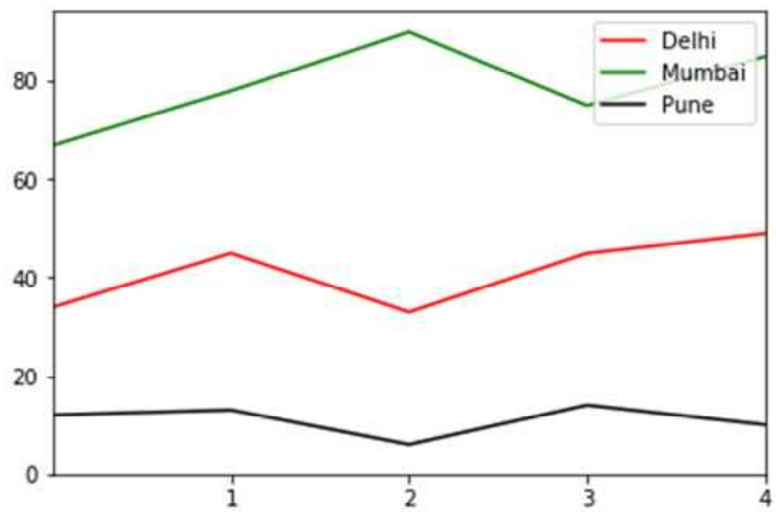
To plot line chart, we can use the following command: xticks and yticks are used to put significant range on axis.

```
sales.plot(xticks=range(1,5),yticks=range(0,100,20))
```



We can define colors for different lines:

```
#Define color options
colors = ['Red','Green','Black']
sales.plot(xticks=range(1,5),yticks=range(0,100,20),color=colors)
```

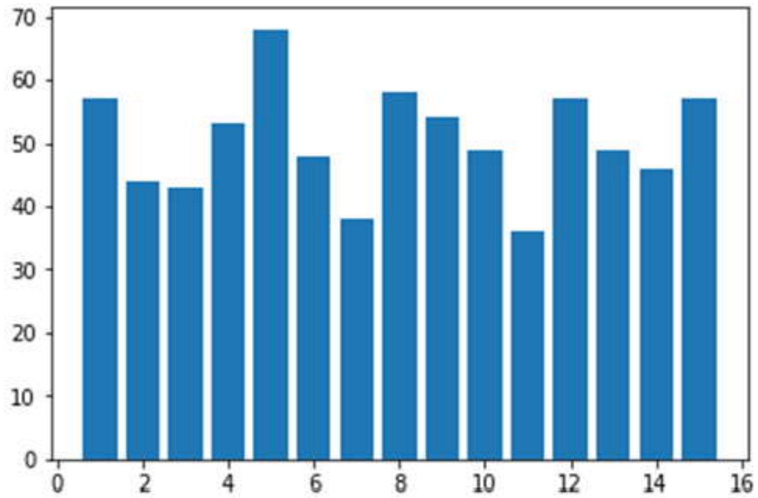


4.2 Bar Chart

Bar chart is used to analyze the grouped data. It is used with categorical data to visualize the impact in each category of other variable. It creates a rectangle for each category. For example, we have three categories for a city and want to see number of vehicles in these cities. By using bar chart, we can generate three rectangles (each representing a city) and height shows number of vehicles. It shows number of observations in each category.

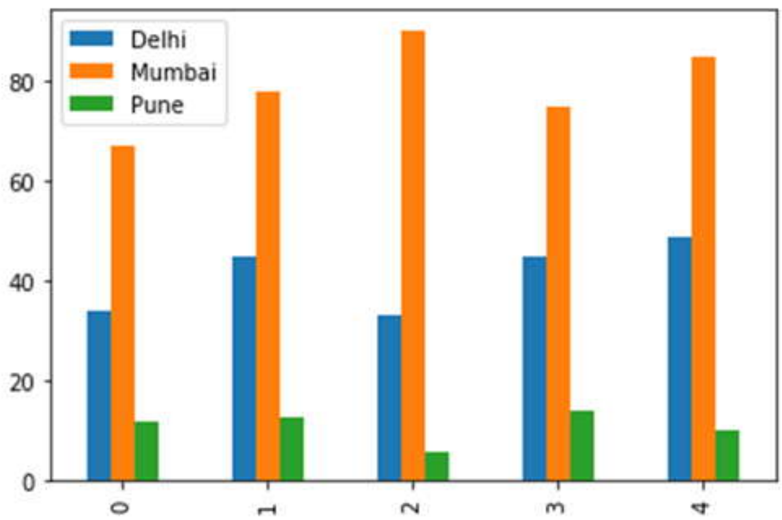
We can plot bar chart by using following code:

```
plt.bar(a,b)
```



We can also generate bar chart from pandas DataFrame. By default, plot function generates line chart. We can instruct the function to generate bar chart.

```
sales.plot(kind='bar')
```

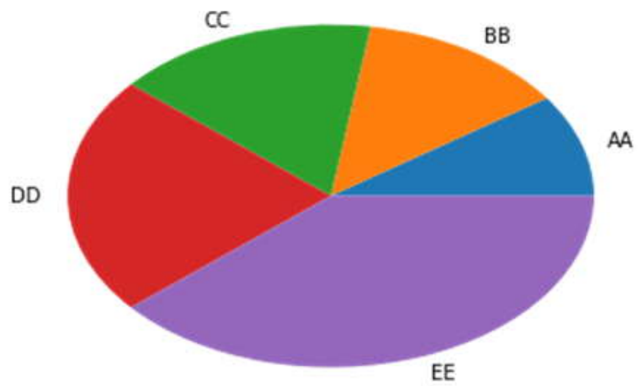


4.3 Pie Chart

Pie chart represent whole data as a circle. Different categories make slices along the circle based on their proportions. For example, the analysis of vehicles in three cities, we use pie chart, which will present proportion of the vehicles in a city as compare to others. Pie chart is useful if we want to compare each category in a common scale.

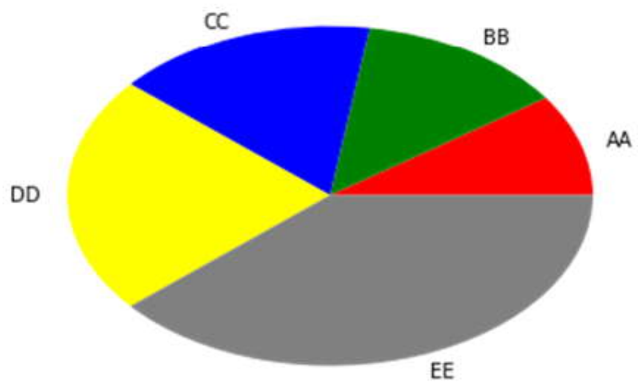
We can generate pie chart by using the following command:

```
a = [3,4,5,7,12]
plt.pie(a,labels=['AA','BB','CC','DD','EE'])
```



We can define color for each slice by:

```
#We can define colors for each category
color_list = ['Red','Green','Blue','Yellow','Grey']
plt.pie(a,labels=['AA','BB','CC','DD','EE'],colors=color_list)
```



4.4 Histograms

Histograms allows us to determine the shape of continuous data. It is one of the plot which is used widely in statistics. Using histograms, we can detect the distribution of data, outlier in the data and other useful properties.

To construct histograms from continuous data, we first need to create bins and put data in the appropriate bins. These are different from Bar charts which are used with categorical variables.

For example, we have following data:

Data = [23,12,34,56,43,26,13,39,58,32,44]

We can define bins, like this:

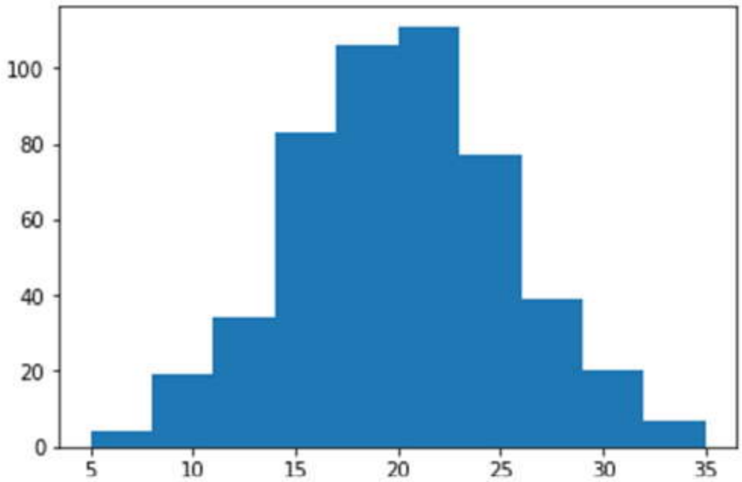
Bin	Data Points	Count
>20	12,13	2
20-40	23,34,26,39,32	5
40-60	56,43,58,44	4

Using matplotlib, we can create histograms:

```
#Generate random integers with mean and sigma
mean = 20
sigma = 5
hist_data = numpy.random.normal(mean, sigma, 500).astype(int)

plt.hist(hist_data)
```

This will generate the following histogram:



4.5 Scatter Plot

Scatter plot is used to show the relationship between two set of data points. For example, any person weight and height, traffic intensity and number of cars, number of paper published and years of experience etc. can be presented by using scatter plot.

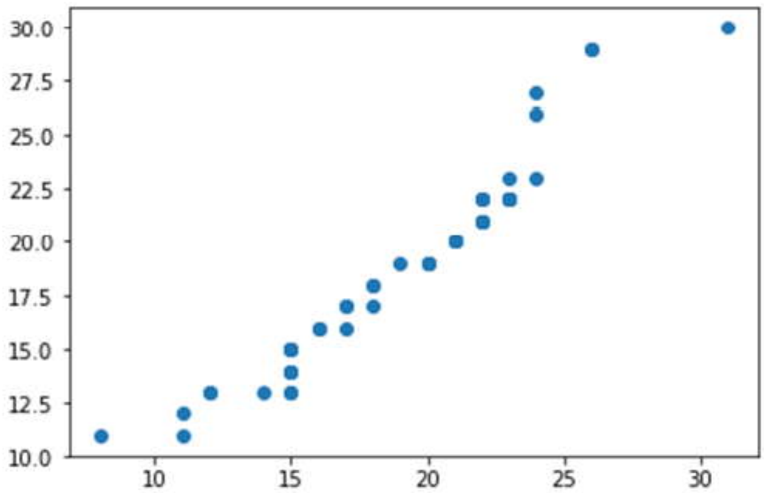
We can use the following code to generate scatter plot:

```
#Generate random integers with mean and sigma
mean = 20
```

```
sigma = 5

#Generate the data and sort it
scatter_data_1 = numpy.sort(numpy.random.normal(mean, sigma,
50).astype(int))
scatter_data_2 = numpy.sort(numpy.random.normal(mean, sigma,
50).astype(int))

#Scatter plot
plt.scatter(scatter_data_1,scatter_data_2)
```



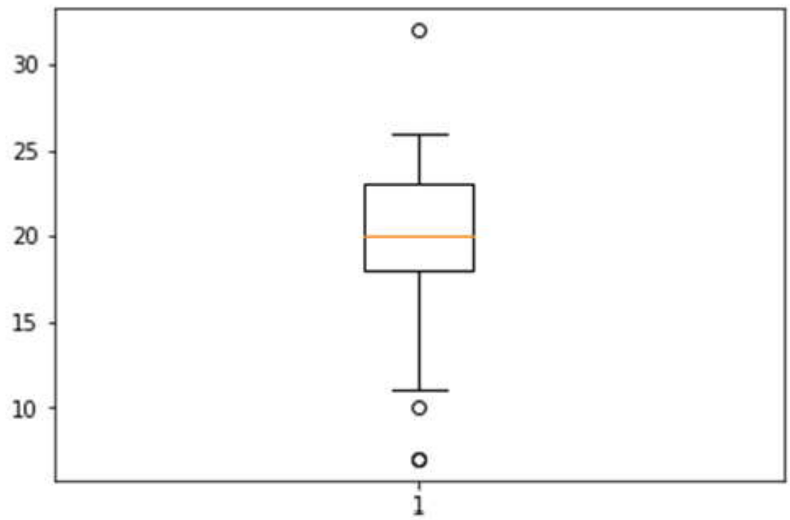
4.6 Box Plot

It is used to understand the variable spread. In a box plot, rectangle top boundary represents third quantile, bottom boundary represents first quantile and line in the box indicates median.

Vertical line at the top indicates the maximum value and vertical line at the bottom indicates the minimum value.

We can generate box plots by using the following code:

```
box_data = numpy.random.normal(56, 10, 50).astype(int)
plt.boxplot(scatter_data_1)
```



4.7 Plotting using Object Oriented way

In above examples, we have used functions to generate the plots. We can also use object-oriented way to generate this graphics. We first define a figure object and keep adding plot elements to enrich it.

First, we create a blank figure object and add axes to it. Then we generate plot within the object and specify parameters for the plot.

Here is the code for this method:

```
#On one axis number from 1 to 15
a = range (1,16)

#On other axis generate random integers with mean and sigma
mean = 50
sigma = 10
b = numpy.random.normal(mean, sigma, 15).astype(int)

#First, we define figure object
figure_object = plt.figure()

#Add axes
axes = figure_object.add_axes([.1,.1,1,1])

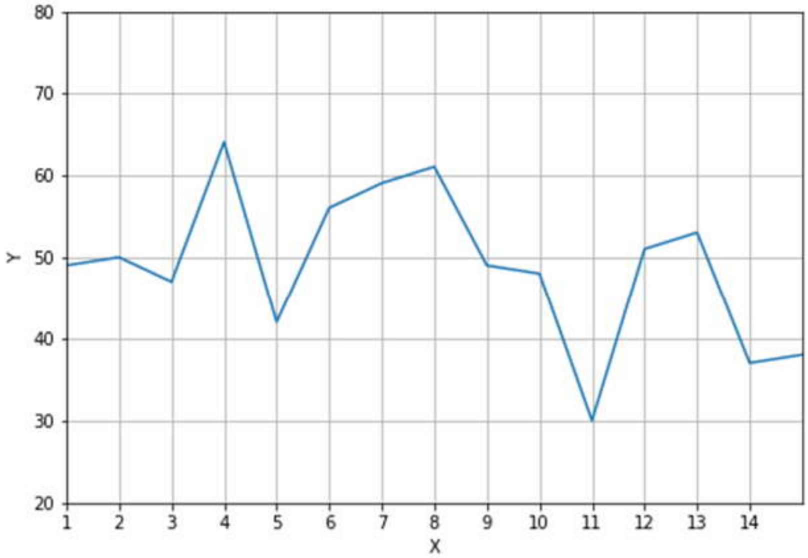
#Adding grid
axes.grid()

#Set axes labels
axes.set_xlabel('X')
axes.set_ylabel('Y')
```

```
#Set axes ticks
axes.set_xticks(range(1,15))
axes.set_yticks(range(20,100,10))

#Set axes limit
axes.set_xlim([1,15])
axes.set_ylim([20,80])

#Generating plot
axes.plot(a,b)
```



In above code,

1. We have created figure object by using plt.figure().
2. Axes are defined by using add_axes method.
3. Grid is enabled by grid() method
4. Labels for the axes are defined.
5. Values on axes (xticks, yticks) are defined.
6. Axes lower and upper limit is added.
7. Plot is generated by using plot method.

We can also create sub plots. Using sub plots we can compare plots by:

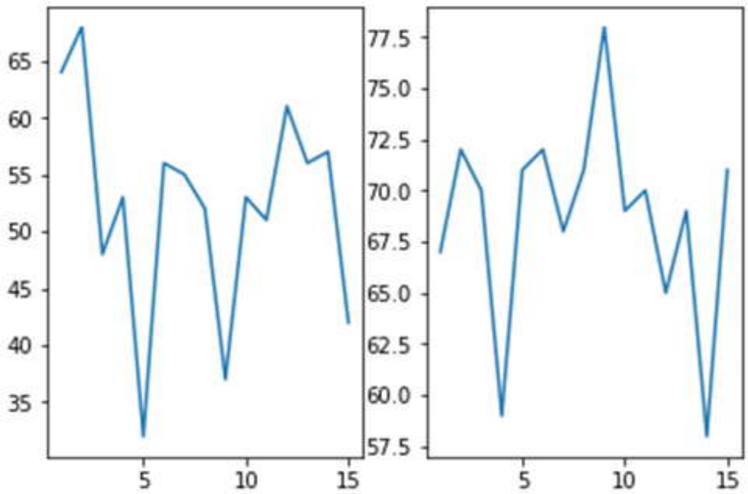
```
mean = 20
sigma = 5
c = numpy.random.normal(mean, sigma, 15).astype(int)

#Create figure object
fig_sub_object = plt.figure()
```

```
#Two axes inside figure object.
number_of_rows= 1
number_of_cols = 2
fig_sub_object, (axes1,axes2) =
plt.subplots(number_of_rows,number_of_cols)

axes1.plot(a,b)
axes2.plot(a,c)
```

This will generate plot like this:



4.8 Seaborn

Seaborn is a python package based on matplotlib. It provides some high level graphical methods for statistics. We will see some of the common plots of seaborn. First import seaborn:

```
import seaborn
```

Create some random data:

```
#Generate random integers with mean and sigma
mean = 25
sigma = 10
dist_data_1 = numpy.random.normal(mean, sigma,
500).astype(int)
dist_data_2 = numpy.random.normal(mean+5, sigma-4,
500).astype(int)
dist_data_3 = numpy.random.normal(mean-5, sigma+2,
500).astype(int)
dist_data = pandas.DataFrame({"A"
```

```
:dist_data_1,"B":dist_data_2,"C":dist_data_3})

#First we need to create categorical data
data = ["Delhi", "Pune", "Ajmer"]

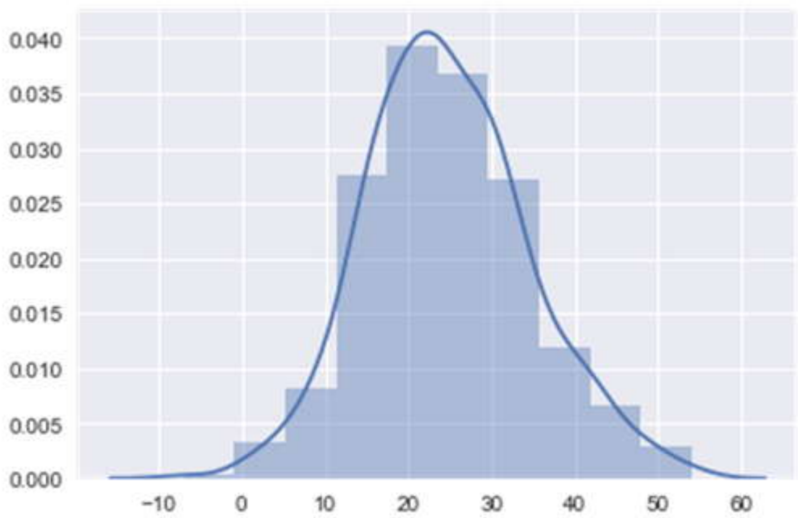
#This code will create categorical data of 500 length with
some predefined probability of each class.
dist_data['city'] = numpy.random.choice(data, 500, p=[0.5,
0.2, 0.3]).tolist()

#Create class variable
dist_data['gender'] = numpy.random.choice(['Male','Female'],
500, p=[0.6, 0.4]).tolist()
```

4.8.1 Distplot

Distplot is useful for exploring univariate distribution. We can build a histogram and put kernel density estimation on it. Kernel density estimation is a non-parametric approach to get probability function of a random variable. If we compare kernel density estimators with histogram, they are smooth and do not have any end points.

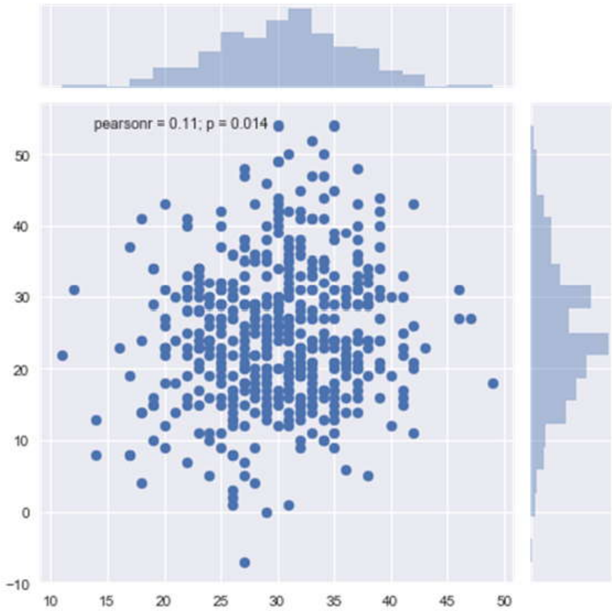
```
seaborn.distplot(dist_data_1,bins=10)
```



4.8.2 Jointplot

it is useful to do bivariate analysis. It is similar to scatterplot in matplotlib.

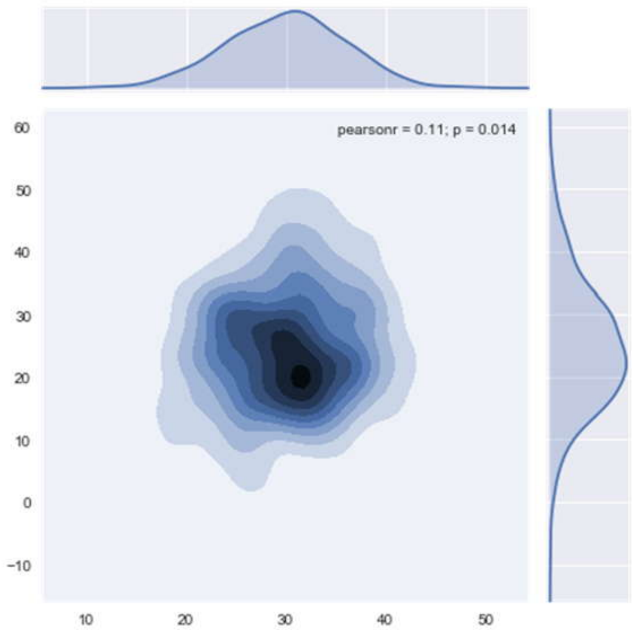
```
seaborn.jointplot(x=dist_data_2, y=dist_data_1);
```



4.8.3 Kernel Density Estimation for bivariate distribution

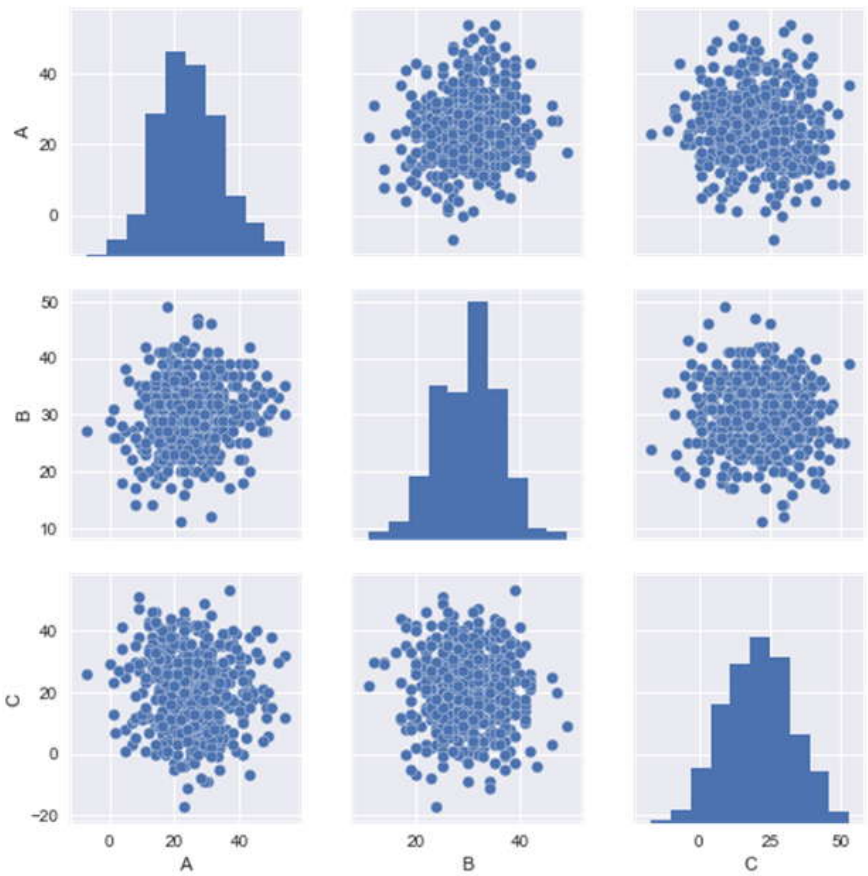
We can use the same jointplot but provide a parameter kind='kde' to perform the kernel density estimation.

```
seaborn.jointplot(x=dist_data_2, y=dist_data_1,kind="kde")
```



4.8.4 Bivariate analysis on each pair of features

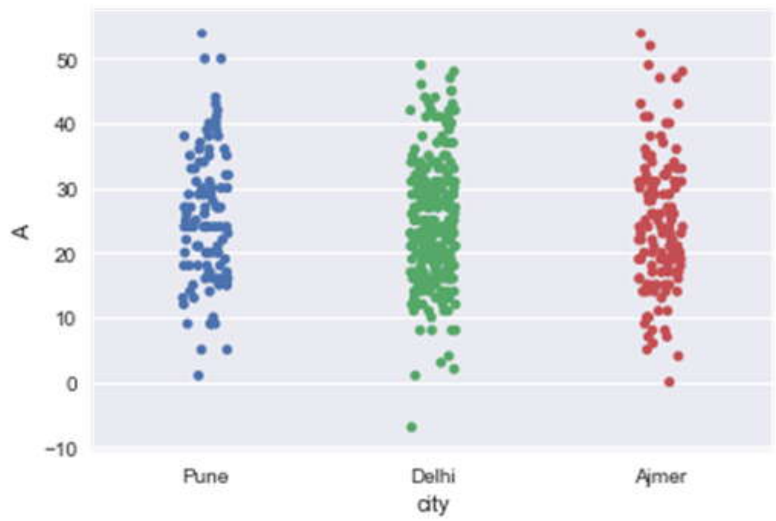
```
seaborn.pairplot(dist_data)
```



4.8.5 Categorical Scatterplot

This is the way of analyzing the categorical variables.

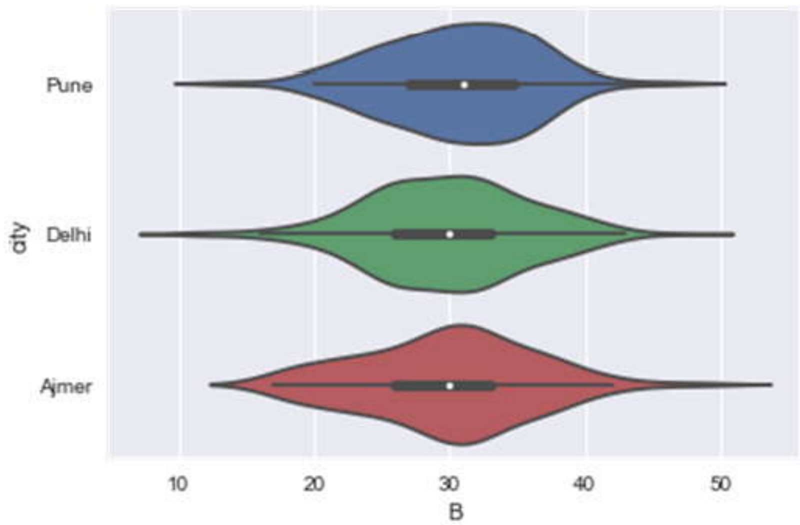
```
seaborn.stripplot(x="city", y="A", data=dist_data,
jitter=True)
```



4.8.6 Violinplots

Scatterplot gives very little information regarding underlying distribution along with categorical variable. Violinplots can be used here to analyze the data. These are boxplot with kernel density estimations.

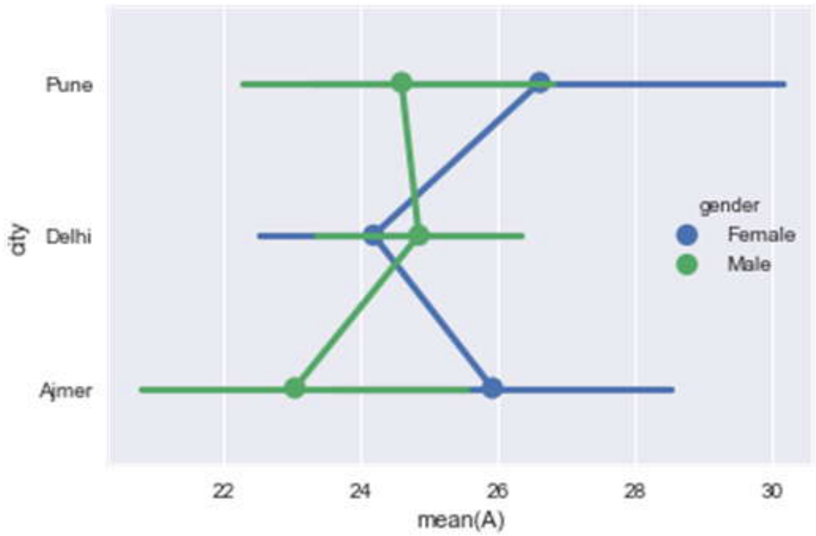
```
seaborn.violinplot(x="B", y="city", data=dist_data)
```



4.8.7 Point Plots

It is used to show the height along with axis. It's class variable is available which connects them.

```
seaborn.pointplot(x="A", y="city", hue='gender',
data=dist_data)
```



Chapter 5

Regression

Regression is a technique of predicting the continuous valued output from input. Input data is given in the form of features. Output or response variable is continuous. In regression, we find model that maps these features to continuous response variable. So, in a way, our model is learning a relationship between input and output.

Let's understand the problem of regression by these examples:

1. **Predicting Stock price:** In stock market, stock price goes up and down depending on the different factors. We can build a regression model that takes input of news related to that stock, its price window for last one-month, social media follower's activity, market position, peer performance last month and predict the price of the stock in next week.
2. **Forecast sales of a month:** In business, it is very useful to predict sales and do the necessary actions. If the prediction of sales is lower, we need to improve some of our offering (may be better marketing or after sales service) and if prediction is high, we need to manage our resources accordingly.
3. **Predict airfare:** During holidays and long weekends, everyone wants to travel. Flight companies also tries to make most of the revenue at that time. Also, it depends on season. For example, during holidays in summers, people stays at home. But during winter and rainy season they want to go somewhere which is away from high traffic. Airlines predict the price based on season, holiday, festival, number of seats occupied, last year minimum and maximum price etc.
4. **Predict like/comment on social media:** When companies want to put any offer/deals for the customer, it is important to do analysis before posting any stuff publicly. Based on the response on their previous post, they can understand and predict likes/comments on new post and can modify them to attract more and more audience.

We first start with simple regression. In this part of regression, input and output are single variables. We will try to fit a single line that fits best on the given output. We will use gradient descent algorithm to find the best line that fit the data. Using this algorithm, we will get intercept and slope of the line.

After this we will move to more complex form of regression, which is multiple regression. Here, we consider many input variables to predict the outcome.

Measuring the performance of our model is important. Sometime complex model

looks great on training but perform worst on test or new data. We will define different type of error and tries to minimize the desired error. We will also learn about the bias variance tradeoff.

As the complexity of model increases we get better performance on training data. But it starts performing bad on test data. We will learn ridge and lasso regression in next chapter that deals with this problem. Lasso regression also used for feature selection. We will also learn some other regression techniques like kernel regression.

5.1 Simple Regression

As the name suggest, it is simple. Given only one input, we must fit a line for output.

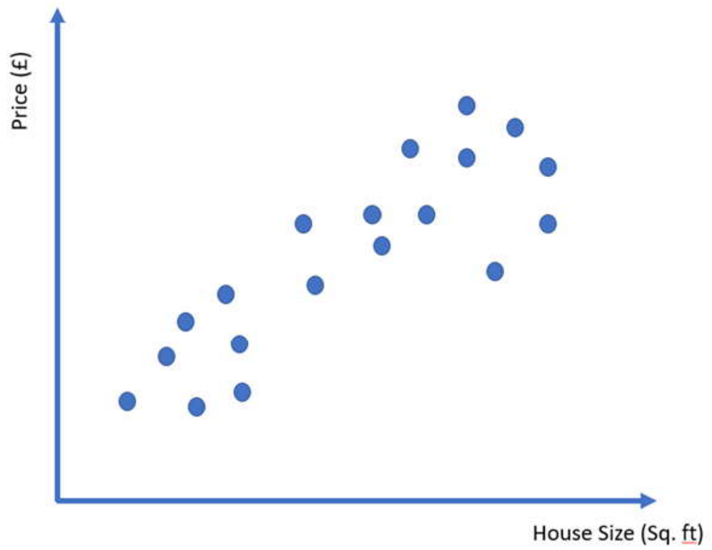
We will understand this approach by predicting the price of the house. To do this, we have past data related to the house price.

Let's say we have two data points, size of the house in square feet and price of the house. Size of the house is considered as input which we give to our algorithm and price is considered as output.

So, our data is in two points (x, y).

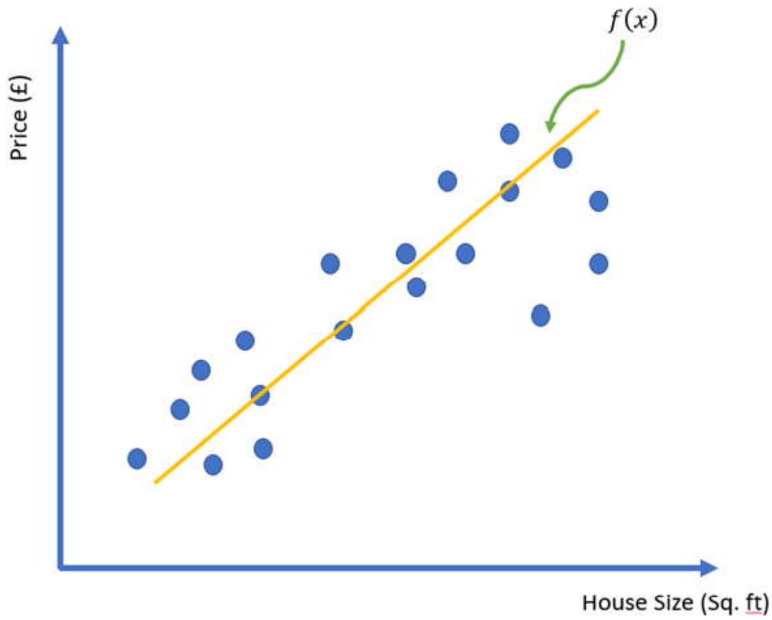
(x, y) -> (size of house, price of the house)

Let's draw the points on the plane.



For any point draw a perpendicular to x axis. That represent house of size. Also draw a perpendicular to y axis. That represent price of the house.

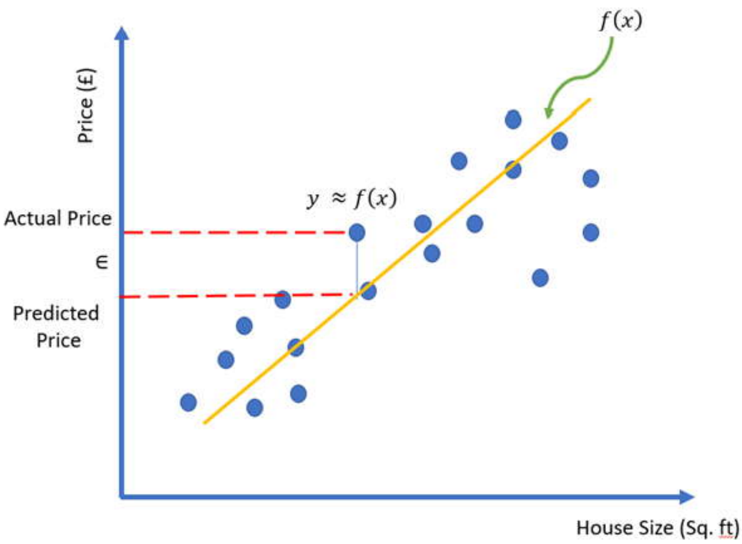
Let's draw a line that maps the relationship between house size and price.



f(*x*) is a function that denotes the expected relationship between the house size and price.

Consider a point. According to our relationship function, house price is approximately equal to f(house size).

$$y \approx f(x)$$



So, the value of house i is given as:

$$y_i = f(x_i) + \in_i$$

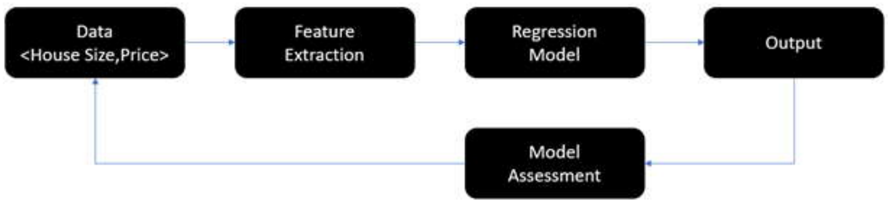
$\in_i$ is given as error in prediction for i[th] house. This error is positive, negative or zero for different examples. Expected value for error is zero.

$$E[\in_i] = 0$$

So, our y_i can be above or below the line equally likely.

In regression we must do two tasks. First task is to select the type of function for the given problem. Is the function constant with output? Or is it linear, quadratic, cubic or any higher order? After choosing the type of function, we must find the estimated function for the given problem.

Our overall regression model can be explained by this:



Our data for this problem is house size and price. In feature extraction, we do data cleaning (handle missing data, outliers etc.). After we build our regression model and generate the output. Now, our output is compared with the actual output in model assessment. This process keeps iterating till the function gives best approximation.

In Simple regression we have to fit a line. Equation of our fitted line can be given as:

$$f(x) = \beta_0 + \beta_1 x$$

The prediction value or output is given by:

$$y_i = \beta_0 + \beta_1 x_i + \in_i$$

Here, $\in_i$ refers to the error term or distance of true value from the predicted value.

β_1 and β_0 are coefficient of our regression model which is called slope and intercept.

Let's understand the cost of line in regression model. This is measured by using Residual Sum of Square or RSS. RSS is the sum of squared difference of actual value and the predicted value.

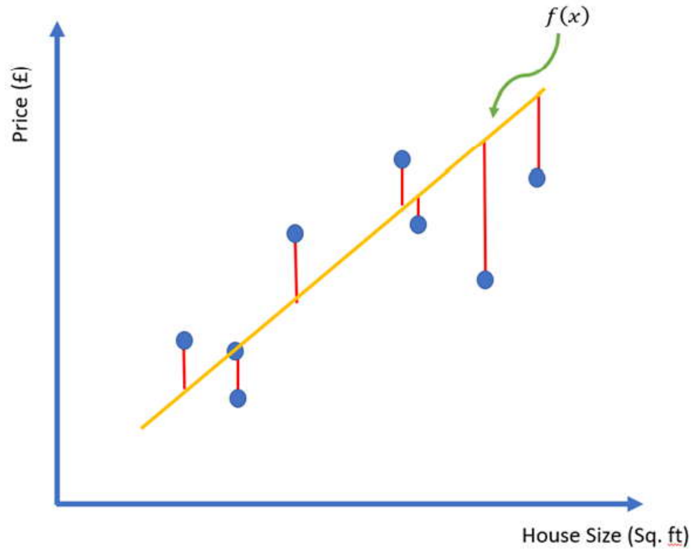
RSS = $(\text{Predicted Price} - \text{Actual Price})_1^2$ + $(\text{Predicted Price} - \text{Actual Price})_2^2$ + $(\text{Predicted Price} - \text{Actual Price})_3^2$ + .. + $(\text{Predicted Price} - \text{Actual Price})_n^2$

Consider the equation in input x, output y and parameters β_1 and β_0.

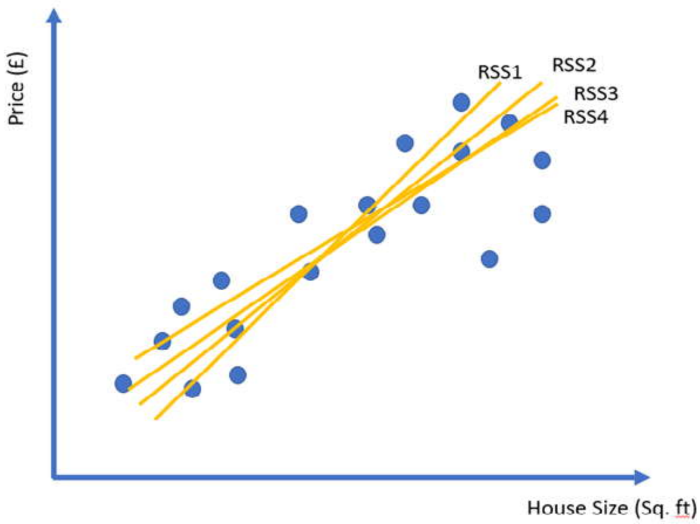
$$RSS = (y_1 - (\beta_0 + \beta_1 x_1))^2 + (y_2 - (\beta_0 + \beta_1 x_2))^2$$
$$+ (y_3 - (\beta_0 + \beta_1 x_3))^2 + \dots + (y_n - (\beta_0 + \beta_1 x_n))^2$$

$$RSS = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2$$

Here, n is the total number of examples in training.



So, our goal for regression is to minimize the RSS. Our algorithm should select the line which gives minimum RSS on the training data.



Once we finalize the line or relationship, we can just put the values in the function and get the output.

We try to find the estimated parameters $\hat{\beta}_1$ and $\hat{\beta}_0$. Estimated parameters are our best guess to actual parameters. So, for any input x we make the prediction for y by using this:

$$\hat{y}_n = \hat{\beta}_0 + \hat{\beta}_1 x_n$$

Here, error may exist in calculating output y, but that is known. That may be positive or negative. So, our best guess is to output the value on the line.

Suppose. the value of $\hat{\beta}_1$ is 450 and $\hat{\beta}_0$ is 20000. Now, Size of any house is 800 Square feet. Now, what would be the price of that house as per our regression model?

Price = 450 × 800 × 20000

Price = £380,000

We can also do in reverse way. Suppose we have £600,000 and we want to know what is the approximate size of house, through which we can get with this amount. With the same equation and parameters:

£600,000 = 450 × House Size + 20000

House Size $\approx$ 1288.88 Sq. Ft.

So, the equation of simple regression works in both ways.

$\hat{\beta}_0$ represents intercept of the line. $\hat{\beta}_1$ represent slope of the line. Slope means when there is a unit change in x, what is the change in output y.

House price with area 100 – House price with area 99 = Price per square feet

$$= (\hat{\beta}_0 + \hat{\beta}_1(100)) - (\hat{\beta}_0 + \hat{\beta}_1(99))$$

$$= \hat{\beta}_1 \text{ (Slope of the line)}$$

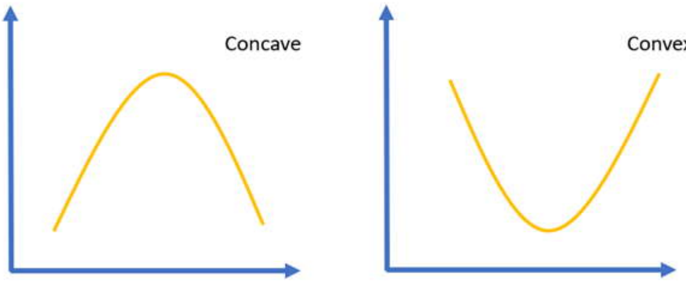
Now, target the top question. How to find the best line. We have calculated RSS by using this formula.

$$\text{RSS} = \sum_{i=1}^{n} (y_n - (\hat{\beta}_0 + \hat{\beta}_1 x_n))^2$$

Here, x is the input and y is the output. That will remain same. What we can do is change the value of parameters $\hat{\beta}_1$ and $\hat{\beta}_0$ to reduce the RSS. So, our objective is

$$\min_{\beta_1,\beta_0} \sum_{i=1}^{n} (y_n - (\beta_1 x_n + \beta_0))^2$$

First, we ne ed to understand the concept of concave and convex function.



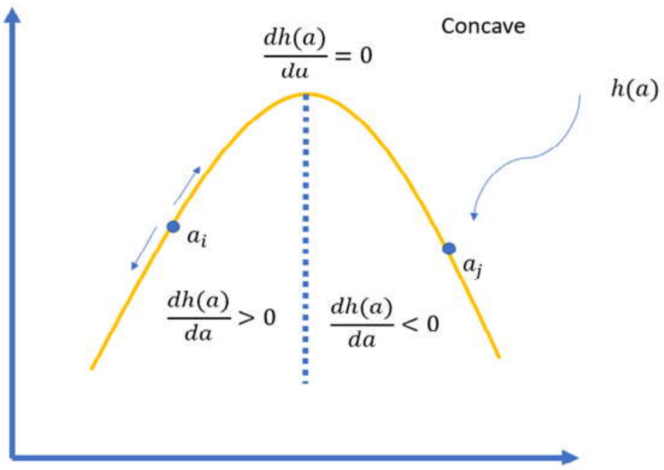
- **Concave function:** These are the function on which if we choose two points and apply function, and draw a line between those two points, they always remain below the function.
- **Convex function:** These are the function on which if we choose two points and apply function, and draw a line between those two points, they always remain above the function.

In Concave or convex function, derivative is zero at one point only. If derivative is not zero anywhere or zero at multiple location, function is neither concave nor convex. So, by using derivative we can find maximum or minimum of the function.

Maximum value of a concave function can be get by hill climbing approach. Hill climb approach starts with one point and move towards the solution closer. Consider a point 'a' on the curve. We want to find the point where function value is maximum.

Let's take the point 'a' in the function h(a). From point 'a' we move to the right or left direction. We can take the derivative of the function h(a).

At point a_i derivative of the function is positive, so we can increase it from 'a'. At point a_j derivative is negative, so we want to decrease 'a'. Our goal is to reach to a point where slope (derivative) is zero that is at the maximum. We can divide in two regions. One is where the derivative is positive, and another is where the derivative is negative. At the point of maximum value derivative is zero.



The point where derivative is zero is the point of convergence. So, we can move the point like this:

$$a_{k+1} = a_k + \frac{dh(a)}{da}$$

If the derivative is positive a_{k+1} is greater than a_k. If the derivative is negative a_{k+1} is smaller than a_k. If derivative is zero both a_{k+1} and a_k is same and we reach to maximum.

It is very difficult to get the exact maximum. We can say that if $a_{k+1} \approx a_k$, we stop the algorithm. We can choose the difference between two values and consider them nearly equal.

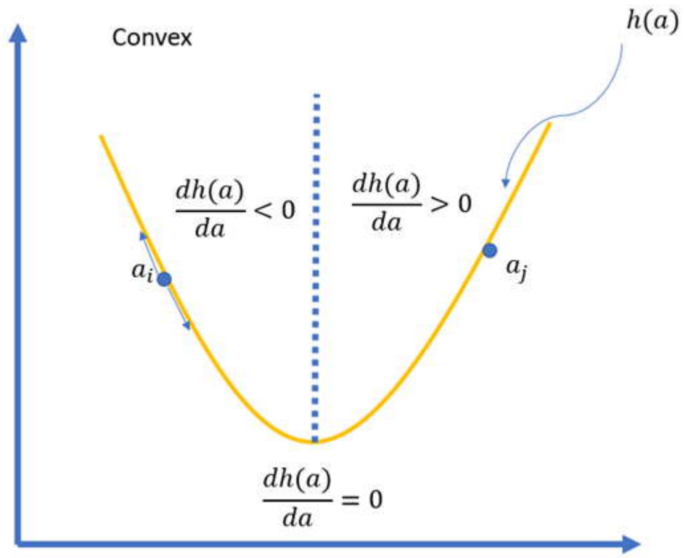
Now to control the speed of updating points, we introduce another parameter learning rate.

$$a_{k+1} = a_k + \gamma \frac{dh(a)}{da}$$

Here, γ referred as a learning rate. Choosing learning rate is very important. If we choose learning rate too high, we may pass the maxima and move around the maxima. If we choose learning rate too low, we may end up taking a lot of steps to reach maxima.

One solution for this problem is dynamic learning rate. Start with higher learning rate and slowly decrease it.

The same way, we find maximum and minimum of convex function.



For convex function, we want to find our minimum. At any point 'a' if the derivative or slope is negative means we have to increase its value to move at the bottom

point. Similarly, if the derivative of any point is greater than zero, it means we have to decrease value of 'a' to come closer to bottom. At bottom derivative is zero, so the update equation is given as

$$a_{k+1} = a_k - \gamma \frac{dh(a)}{da}$$

Remember, the minus (instead of plus) in hill descent approach.

Let's understand our objective of regression problem. We want to find the parameters β_1 and β_0 such that RSS is minimum.

Here, we have more than one parameter. So, we use gradient instead of derivative. We denote the function like this:

$$\nabla h(\beta)$$

Here, ∇ is gradient and β is the set of parameters.

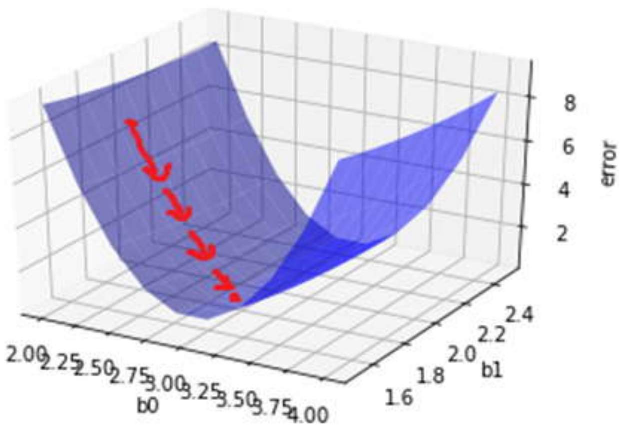
$$\nabla h(\beta) = \begin{bmatrix} \dfrac{\partial h}{\partial \beta_0} \\ \dfrac{\partial h}{\partial \beta_1} \end{bmatrix}$$

Here, $\dfrac{\partial h}{\partial \beta_i}$ is referred as partial derivative of function h with respect to parameter β_i. In partial derivative, we treat all the other parameters as constant and find the derivative with respect to β_i.

Like convex function, we start from a point and move in the direction where gradient is zero. We can write the convergence function like this:

While $\beta_{k+1} \approx \beta_k$

$$\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}_{k+1} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}_k - \gamma \begin{bmatrix} \dfrac{\partial h}{\partial \beta_0} \\ \dfrac{\partial h}{\partial \beta_1} \end{bmatrix}$$



So, to apply regress ion algorithm we assume that solution is unique and gradient descent converges to minimum. Our objective is to minimize RSS.

$$RSS_{minimum} = \min_{\beta_0, \beta_1} \sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 x_i) \right)^2$$

Let's use our gradient approach to get this. We are not going in the calculus of finding partial derivatives. When we apply gradient, we get this:

$$\nabla RSS = \begin{bmatrix} -2\sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 x_i) \right) \\ -2\sum_{i=1}^{n} x_i \left(y_i - (\beta_0 + \beta_1 x_i) \right) \end{bmatrix}$$

We know when gradient is zero we reach to minimum. So, we can take $\nabla RSS = 0$ and find the value for β_1 and β_0. Most of the time there is no possible solution by using this technique.

Another, way to do is gradient descent. We start with some values and keep moving in the direction where RSS is reducing and finally approaching to zero.

While $\beta_{k+1} \approx \beta_k$

$$\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}_{k+1} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}_k - \gamma \begin{bmatrix} -2\sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 x_i) \right) \\ -2\sum_{i=1}^{n} x_i \left(y_i - (\beta_0 + \beta_1 x_i) \right) \end{bmatrix}$$

Rewriting, above equation:

$$\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}_{k+1} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}_k + \gamma \begin{bmatrix} -2\sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 x_i) \right) \\ -2\sum_{i=1}^{n} x_i \left(y_i - (\beta_0 + \beta_1 x_i) \right) \end{bmatrix}$$

So, using above we can calculate the coefficient of simple regression model.

Now, let's understand simple regression to predict the outcome. We will do this in

jupyter notebook. It will give warnings depending on the version of python and libraries you are using. First, we will make it off.

```
import warnings
warnings.filterwarnings('ignore')
```

Import libraries. (It will contain some additional libraries which will be used in other section of this chapter.

```
#Import libraries
import numpy
import pandas
import sklearn
import seaborn
import matplotlib.pyplot as plt
%matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 10, 8

from sklearn import datasets, linear_model
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso,
Ridge
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error
from sklearn.cross_validation import KFold
from sklearn.datasets import load_boston
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from scipy.interpolate import spline
from sklearn.neighbors import KNeighborsRegressor
from sklearn.kernel_ridge import KernelRidge
```

Read the data:

```
regression_data = pandas.read_csv('./data/simple_regression_
data.csv')
```

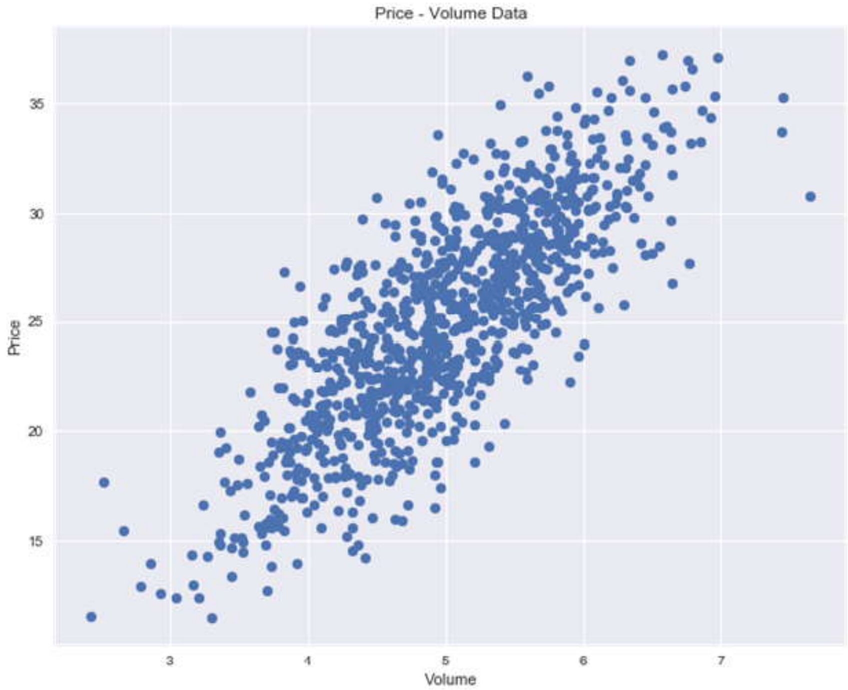
This data contains volume and price of a metal taken by different vendors at different time.

```
regression_data.head()
```

First, we plot the data

```
plt.scatter(regression_data['Volume'],regression_data['Price'])
plt.xlabel('Volume')
```

```
plt.ylabel('Price')
plt.title('Price - Volume Data')
```



Divide the data into training and test set. Training set contains 80% of the data. Test set contains 20% of the data.

```
X_train, X_test, Y_train, Y_test = train_test_split
(regression_data['Volume'], regression_data['Price'],
test_size=0.20)
```

Create model and predict on test data.

```
# Create linear regression object
simple_linear_regression = LinearRegression()

# Train the model using the training sets
X_train = X_train.values
Y_train = Y_train.values
X_test = X_test.values
simple_linear_regression.fit(pandas.DataFrame(X_train),
pandas.DataFrame(Y_train))

# Make predictions using the testing set
Y_pred =
simple_linear_regression.predict(pandas.DataFrame(X_test))
```

Print mean squared error

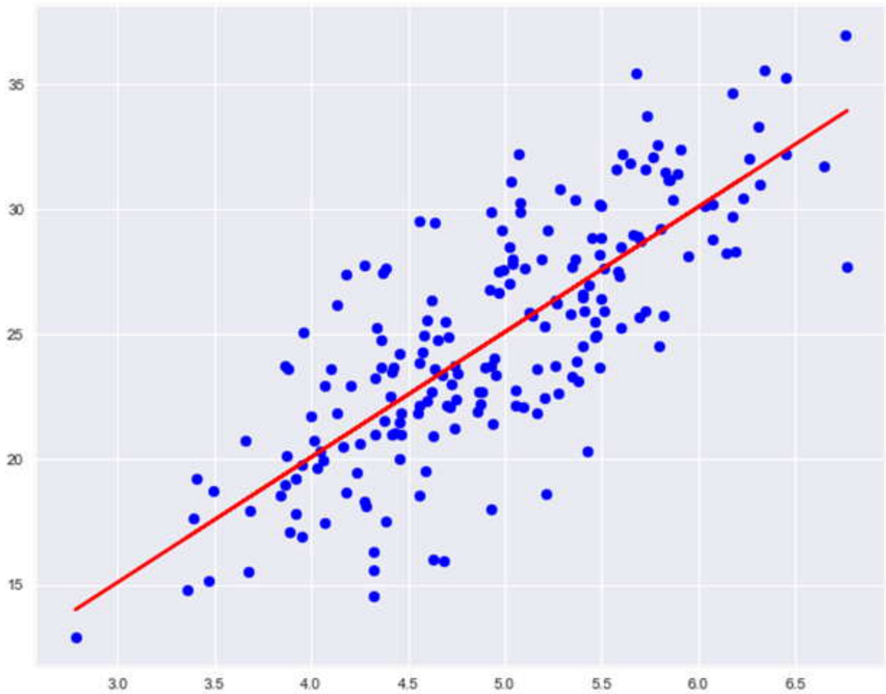
```
print("Mean squared error: %.1f" %
mean_squared_error(Y_test,Y_pred))
```

Print R2 score

```
print('R2 Score: %.2f' % r2_score(Y_test,Y_pred))
```

Plot outputs

```
plt.scatter(X_test, Y_test, color='blue')
plt.plot(X_test, Y_pred, color='red', linewidth=2,)
```



5.2 Multiple Regression

Multiple regression is the linear regression with multiple features. In real world, data sets have multiple features which are equally important to predict output.

5.2.1 Polynomial Regression:

Sometimes output is not with the linear relationship with input. It may relate to input with higher degree of polynomial. This is called **polynomial regression**. General polynomial regression model is given as:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \ldots + \beta_p x_i^p + \epsilon$$

Parameters $= [\beta_0, \beta_1, \beta_3, \ldots \beta_p]$

Features = $[x, x^2, x^3, \ldots x^p]$

In our house price prediction model, we predicted house price with the help of size of the house. There may exists other features like number of rooms, number of bathroom, number of flats in the buildings, crime rate in the area, population of the area etc. So, by using the information of different features we can create better prediction model.

Generalized multiple regression model can be given as:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots + \beta_d x_d + \in$$

$$y_i = \sum_{i=0}^{n} \beta_i x_i + \in$$

Here, x_0 is taken as 1.

Parameters $= [\beta_0, \beta_1, \beta_3, \ldots \beta_p]$

Features = $[x_1, x_2, x_3, \ldots x_d]$

One thing to note, feature is not raw data. They can same or different from input data. For example, consider three inputs house size, number of bathrooms and population. Now, we can form these features from this data:

x_1= house size

x_2 = (number of bathrooms)^2

x_3 = log |population|

We can give the regression model for this problem:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \in$$

We use a line to fit a model relationship between input and output in simple regression. In multiple regression, we need something better. Suppose we have two features and one output. We can consider two inputs along with two axis and output along third axis. So, instead of a line we can draw a separating 2D hyperplane to classify the data. Similarly, for n features, we can create nD plane.

To interpret the model parameters, we can fix the other features and interpret the relationship with output. In our house example let's try to interpret β_1. To do that we need to fix other features of the house. So β_1 is the change in output for unit change in house size when number of bathrooms and population is fixed.

Now, we discuss the algorithm for multiple regression. As the output for one observation is given as:

$$y_i = \sum_{i=0}^{d} \beta_i x_i + \in$$

For considering complete dataset we can rewrite it as:

$$Y = Xw + \in$$

Number of Observations = *n*

Number of features = *d*

Y is a vector of n values. *X* is a matrix of *n* × *d*. w is a vector of *d* values and ∈ is a vector of n values. So,

$$Y_{n \times 1} = X_{n \times d} w_{d \times 1} + \in_{n \times 1}$$

Residual Sum of Squares for multiple regression is defined as:

$$RSS = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_d x_{di}))^2$$

$$RSS = \sum_{i=1}^{n} (y_i - x_i \beta)^2$$

Where β represent vector of *d* length and x_i represent features for *i* example. In matrix form:

$$RSS = (y - X\beta)^T (y - X\beta)$$

*X*β is the predicted value of *y* which is equal to $\hat{y}$

$$RSS = (y - \hat{y})^T (y - \hat{y})$$

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

We can take gradient of RSS as:

$$\nabla RSS = -2X^T (y - X\beta)$$

So, to solve to get ?, take gradient equal to zero.

$$0 = -2X^T (y - X\beta)$$

By solving the above equation, we get:

$$\beta = (X^T X)^{-1} X^T y$$

So, we can get all parameters from solving above. With large number of features, this equation is computationally very expensive to perform.

Let's apply another solution (gradient descent) for this problem. β_{k+1} is the value of parameters at *k* + 1 iteration.

While $\beta_{k+1} \approx \beta_k$

$$\beta_{k+1} = \beta_k - \gamma[-2X^T (y - X\beta_k)]$$

So final equation:

While $\beta_{k+1} \approx \beta_k$

$$\beta_{k+1} = \beta_k + \gamma[2X^T(y - X\beta_k)]$$

In implementation, we can initialize all the parameters to zero.

5.3 Model assessment

So, we have learned the techniques to build regression model. In this section, we will learn the way of model assessment. It is important to assess the model so that you can take consideration of features, model complexity etc.

In our house example, we have data of 100 house. We build our model by using multiple regression and start predicting. But after sometimes, we start feeling that our model is not doing great. On the new large data, our linear relationship is not holding correctly. So, to calculate the loss due to wrong predictions, we use a loss function.

Loss function = $L(y, \hat{f}(X))$

$\hat{f}(X)$ is our predicted value and *y* is the actual value. We have different choice of Loss function.

Absolute Error Loss function = $\left\| y - \hat{f}(X) \right\|$

Squared Error Loss function = $\left(y - \hat{f}(X) \right)^2$

5.3.1 Training Error

Training error is the error of prediction measured on the training set. Training set is the dataset which is used to build the model.

Average Training error = $\dfrac{L(y, \hat{f}(X))}{\text{Training Examples}} = \dfrac{\left(y - \hat{f}(X) \right)^2}{\text{Training Example}}$

Here, the function $\hat{f}(X)$ is formed by considering training data.

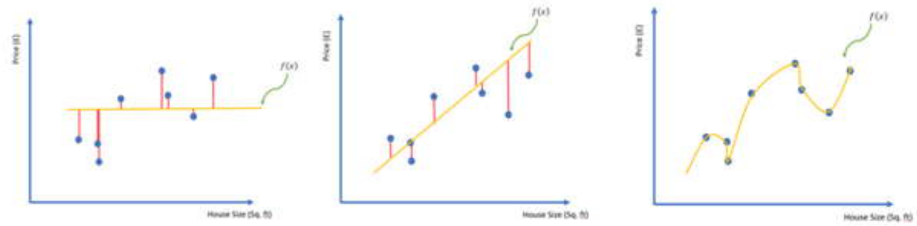
In Statistics and Machine Learning we use another term to define error, Root mean squared error.

$$\sqrt{\frac{\left(y - \hat{f}(X) \right)^2}{\text{Training Example}}}$$

The reason for doing square root is to take the error in the same units of output.

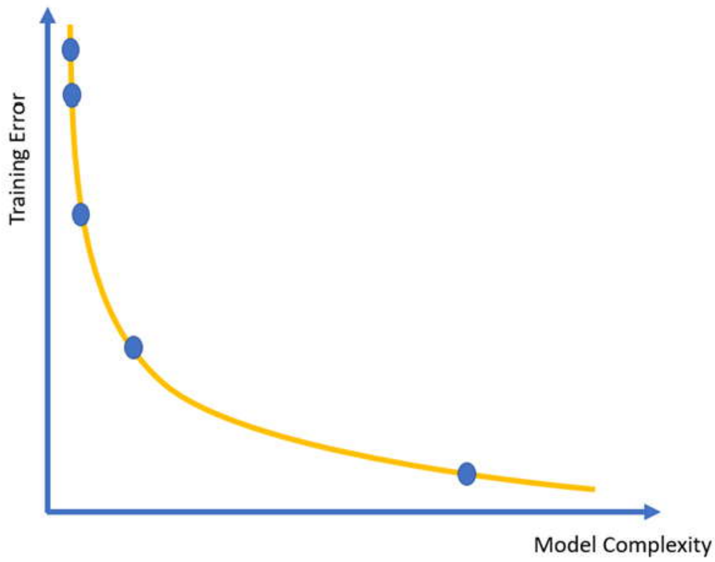
Training error is reduced as the model complexity increases.

Consider these three models:



First is just a point. Giving average result for each input. In this training error is very high. Second is a line. Training error is less than previous model. Third is higher degree polynomial model. Training error is very less in this model.

*



Training error is not a good measure of model performance. It is highly optimistic on training data. We are checking the loss on our trained model and modifying the model accordingly.

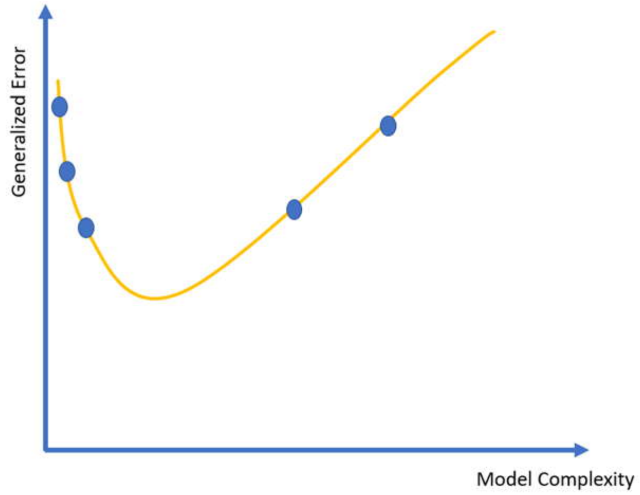
5.3.2 Generalized Error

Generalized error is the error on complete data (and not only training dataset).

This can be given as:

$$E_{xy}\left[L\left(y, f\left(X, \hat{\theta}\right)\right)\right]$$

So, most of the points are not visible during training our regression algorithm. Generalized error does not always decrease with the increasing of model complexity.



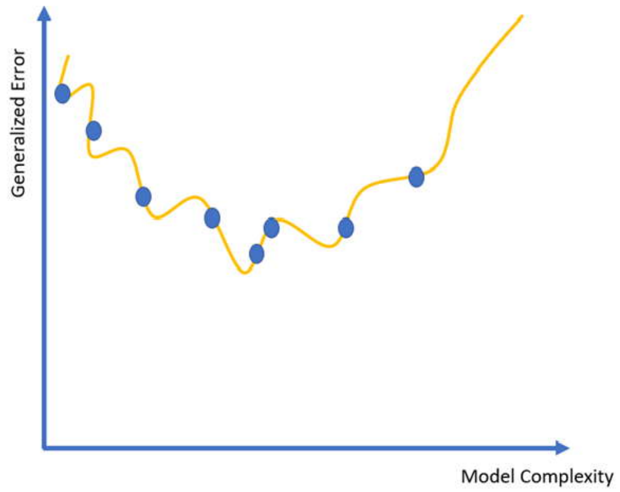
We cannot calculate generalized error. Simple reason is we don't know what is the complete data is. What is coming in future.

5.3.3 Testing Error

We cannot estimate generalization error. So, to get the true error of the model, we use another technique. While creating the model, we hide some example and train the model on remaining example. After the model building is complete, we evaluate the performance on these test set points.

Average Testing error = $\frac{L\left(y, \hat{f}(X)\right)}{\text{Test Examples}} = \frac{\left(y - \hat{f}(X)\right)^2}{\text{Test Example}}$

The benefit of using this error is the model never seen test example. So it is tested on the test point and the error is nearly same as generalized error.



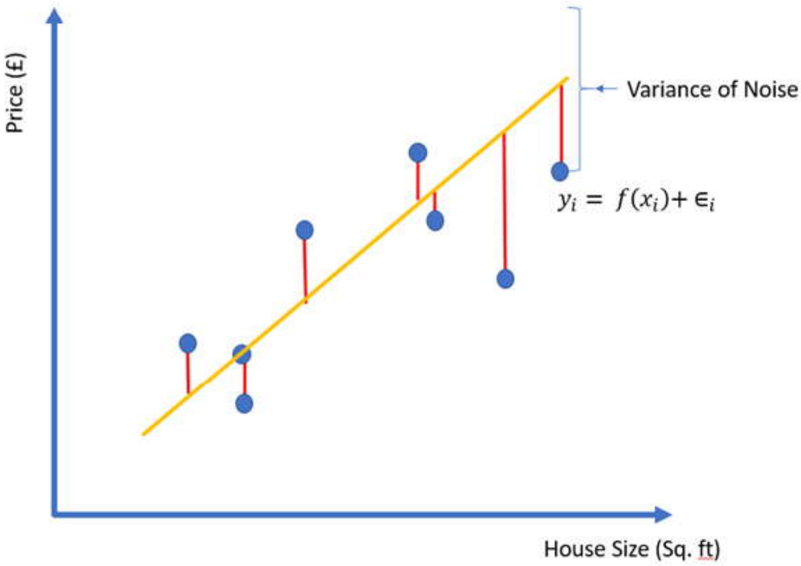
So, as it is clear that Test error is the noisy version of generalization error.

We increase the model complexity and training error is reduced but true error is increase. This is called **overfitting** in machine learning. In general, model fit very well on training data but very bad on true data is prone to overfit.

5.3.4 Irreducible Error

Let say we have found the true relationship *f*(*x*) to predict the output. But the values are still not following the pattern. Consider the example of a house. There may exists two seller having exact same properties, but sellers are selling them in different prices.

So, our prediction cannot be exact in this case. This kind of error or noise we cannot control. This is called as **irreducible error**.



Bias: Bias is the difference between true function and the average fit over the data. If there is bias, there is error. Bias means our average fit does not capture the true relationship.

$$\text{Bias}(\theta) = f_{true}(X) - E\left[\hat{f}(X)\right]$$

Low complexity model has high bias, as they are not close to true relationship. High complexity model has high bias.

Variance: It is the difference of any fit and average fit. In other words, variation between average fit on the data and the fit in consideration.

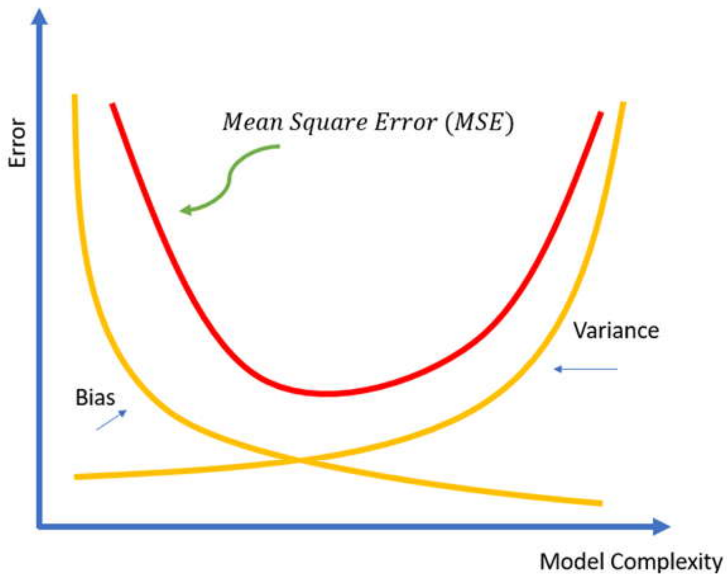
$$\text{Variance}(\theta) = f_i(X) - E\left[\hat{f}(X)\right]$$

If the model complexity is very high variations between any fit and average fit is very low. If the model complexity is low, this variation is low.

5.3.5 Bias-Variance Tradeoff

We can understand the tradeoff between bias and variance by using the curve below. As model complexity is low, bias is high, and variance is low. As the model complexity increases, variance started increasing and bias decreases. Our error depends on both bias and variance. We consider Mean Squared Error (MSE),

$$MSE = (Bias)^2 + Variance$$



We try to choose the point where MSE is least.

We cannot compute bias and variance because they are based on generalized dataset. But we can optimize the tradeoff between these two.

In regression we have to complete two tasks:

- Select the complexity of model
- Compute test error that is near to generalization error.

One way to do this is to divide the dataset in two: Training set and Test set.



First choose some model complexity. This refers to degree of polynomial. Now train the model by using this. Assess the model on the test set. If the performance is not satisfactory, try with other model complexity and retrain the model. Do this step till we get good result or best result in test set.

There is one problem with this approach. Our goal is to find generalization error by using test set. But here we are choosing model complexity in a way that it reduces error on test set. So, in other words we are using information from test set during train which is not allowed. To handle this situation, create another test set. We call it Validation set.



Now train the model by using training set data and evaluate performance with chosen model complexity on validation set. Do it repeatedly till we get best performance on validation set. Now finally test your model on test set. As the data in the test set never used for any training purpose it gives correct measure for generalization error.

There are different type of division of data, which can be done as per the dataset. We can divide the data in 50%, 20%,30% as training set, validation set and test set. In some case we can choose 80%, 10%, 10% as training, validation and test set.

Now will work on python code to perform multiple regression.

In this task we will use Boston housing data and perform multiple regression. This dataset is a part of StatLib Library maintained at Carnegie Mellon University. It is created by Harrison, D. and Rubinfeld, D.L.,J. Environ. This dataset contains 13 continuous attributes and one binary class attribute. Total 506 instances are present.

Scikit learn contains this dataset by default. We will use that.

First, load the data from scikit-learn

```
boston_data = load_boston()
```

boston_data is a dictionary. First check the keys.

```
boston_data.keys()
```

We can get the description of boston_data using DESCR.

```
boston_data.DESCR
```

Here, is the description of the data

- **CRIM:** crime rate per capita
- **ZN:** proportion of residential land zoned
- **INDUS:** proportion of non-retail business acres
- **CHAS:** binary variable. 1 for tract bounds river and 0 otherwise
- **NOX:** nitric oxides concentration

- **RM:** average number of rooms
- **AGE:** owner occupied units
- **DIS:** weighted distance to employment centers.
- **RAD:** index of accessibility
- **TAX:** full value property tax rate
- **PTRATIO:** pupil-teacher ratio
- **B:** proportion of blacks
- **LSTAT:** lower status of population
- **MEDV:** owner occupied homes

Get the attributes or features of the data

```
boston_data.feature_names
```

We have the data and respective column names. We can convert this to pandas DataFrame.

```
boston_data_frame =
pandas.DataFrame(boston_data.data,columns=boston_data.feature_names)
```

Now, we have the data. We will do some feature engineering on it and apply regression techniques.

First, check for missing information.

```
boston_data_frame.isnull().sum()
```

We can separate dependent and independent variables

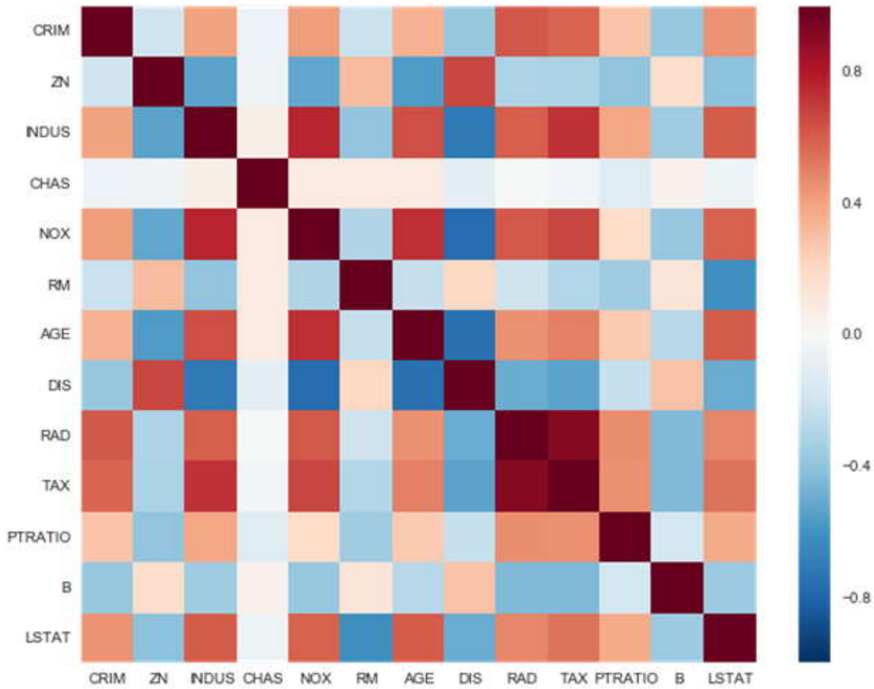
```
boston_data_X =
boston_data_frame[boston_data_frame.columns[0:13]]
boston_data_Y =
boston_data_frame[boston_data_frame.columns[13:14]]
```

Check the correlation

```
boston_data_X.corr()
```

Visualize correlation between attributes by using heatmap

```
seaborn.heatmap(boston_data_X.corr())
```



Feature contains high correlation. We need to remove them first before applying regression techniques.

```
# Create correlation matrix
abs_corr_matrix = boston_data_X.corr().abs()

# Select upper triangle of matrix
up_tri                                                   =
abs_corr_matrix.where(numpy.triu(numpy.ones(abs_corr_matrix.shape),
k=1).astype(numpy.bool))

# Find all the features which is having correlation > 0.75
with other features.
correlated_features = [column for column in up_tri.columns
if any(up_tri[column] > 0.75)]

#Print correlated_features
print(correlated_features)
```

Drop correlated features:

```
boston_data_X  =  boston_data_X.drop(correlated_features,
axis=1)
```

Divide the data into training and test set. Train set contains 80% of the data. Test set contains 20% of the data.

```
X_train, X_test, Y_train, Y_test = train_test_split
(boston_data_X, boston_data_Y, test_size=0.20)
```

Create object of multiple linear regression:

```
linear_regression = LinearRegression()
```

Fit the model:

```
linear_regression.fit(X_train,Y_train)
```

Make prediction on test data:

```
Y_pred = linear_regression.predict(X_test)
```

Mean squared error:

```
print("Mean squared error: %.1f" % mean_squared_error
(Y_test,Y_pred))
```

R^2 score:

```
print('R2 Score: %.2f' % r2_score(Y_test,Y_pred))
```

Mean absolute error:

```
print('Mean absolute error: %.2f' % mean_absolute_error
Y_test,Y_pred))
```

Compare actual value and predicted value:

```
Y_test['Pred'] = Y_pred
Y_test['Difference'] = abs(Y_test['Pred'] - Y_test['Price'])
Y_test.head()
```

We can calculate error by using K-fold cross validation:

```
kfold = KFold(len(boston_data_frame),n_folds=10,shuffle=True)
mean_abs_errors = list()
for train,test in kfold:
    linear_regression.fit(boston_data_X.ix[train],
    boston_data_Y.ix[train])
    Y_test = boston_data_Y.ix[test]
    Y_pred = linear_regression.predict(boston_data_X.ix[test])
    mean_abs_errors.append(mean_absolute_error
    (Y_test,Y_pred))
print('10 Fold Cross validation Error',numpy.mean
(mean_abs_errors))
```

Chapter 6

More on Regression

6.1 Introduction

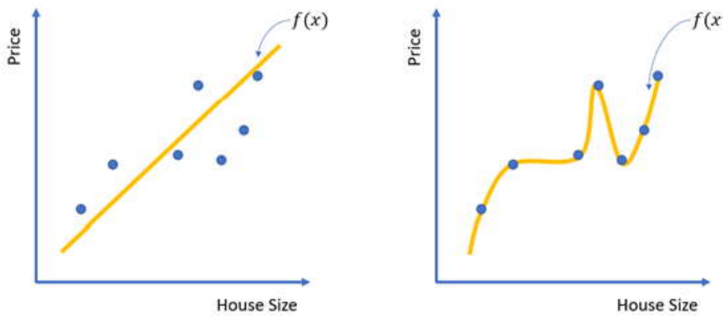
From the discussion of bias and variance, we know that if a model complexity is very high, bias is low, but variance is high. And if the model complexity is low, bias is high, but variance is low. Bias and variance play role in error in the model.

Also, we have learned about overfitting in the data. Overfitting occurs when there exist two model A and B such that:

Training Error (A) < Training Error (B) but,

Generalization Error (A) > Generalization Error(B).

For example, consider two models in the given data points.



First model does not perform very well on training data. Second model produce no error on training data. But this model gives high error on unseen test data. This situation is called overfitting. If we focus more on reducing training error with increasing model complexity, we may end up of overfitted model which gives less performance on test data.

Let's understand increasing model complexity and performance using python code.

Load libraries

```
import numpy
import pandas
import sklearn
import seaborn
import matplotlib.pyplot as plt
%matplotlib inline
```

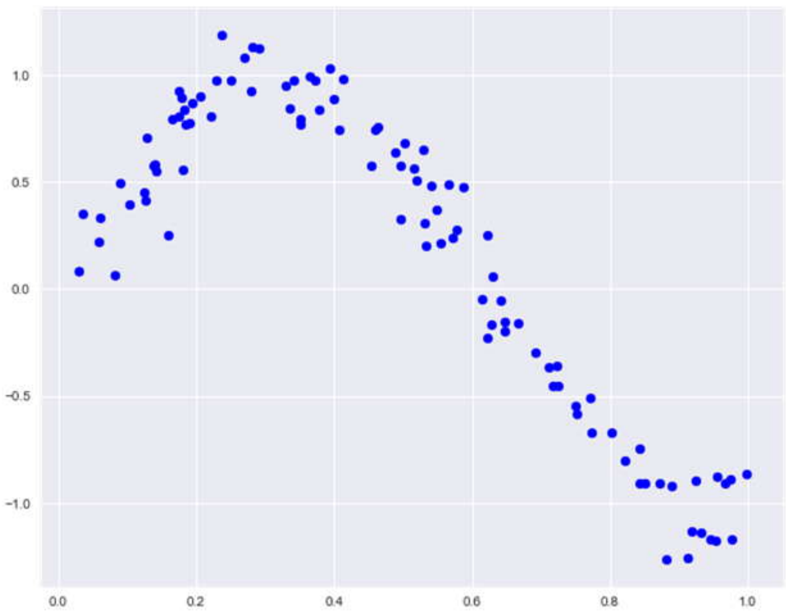
```
from pylab import rcParams
rcParams['figure.figsize'] = 10, 8
from sklearn import datasets, linear_model
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso,
Ridge
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error
from sklearn.cross_validation import KFold
from sklearn.datasets import load_boston
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from scipy.interpolate import spline
from sklearn.neighbors import KNeighborsRegressor
from sklearn.kernel_ridge import KernelRidge
```

Load the data

```
overfitting_data = pandas.read_csv('./data/overfitting_data.
csv')
overfitting_data.head()
```

First plot the data

```
plt.scatter(overfitting_data['X'], overfitting_data['Y'],
color='blue',data=overfitting_data)
```



Divide the data into training and test set. train set contains 70% of the data. test set contains 30% of the data.

```
X = overfitting_data['X']
Y = overfitting_data['Y']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.30)
```

We will use different degree of polynomial on this data and check the result.

Divide the data into training and test set. train set contains 70% of the data. test set contains 30% of the data.

```
#First create a function that take number of degree, train
and test data and generate regression curve.
def  linear_regression_with_degree(X_train,Y_train,X_test,
Y_test,degree):

    #Make pipeline for higher degree feature
    model = make_pipeline(PolynomialFeatures(degree),
    LinearRegression())
    model.fit(pandas.DataFrame(X_train), pandas.DataFrame
    Y_train))

    #Change training and test data as requirement
    X_test = pandas.DataFrame(X_test,columns=['X'])
    X_test = X_test.sort_values(by=['X'])
    Y_test = pandas.DataFrame(Y_test)
    Y_test = Y_test.ix[X_test.index]

    #Generate predictions
    Y_pred = model.predict(X_test)
    #Perform spline smoothing. Higher degree lines will be
    much smooth.
    smooth_feature = numpy.linspace(numpy.min(X_test['X'].
    tolist()),numpy.max(X_test['X'].tolist()),1000)
    smooth_points = spline(X_test['X'].tolist(), Y_pred,
    smooth_feature)

    #Plot the curve
    plt.scatter(X_test, Y_test, color='blue')
    plt.plot(smooth_feature,smooth_points,'-g')
    plt.title('Mean absolute error: %.4f' % mean_absolute_error
    (Y_test,Y_pred) )
```

Function calculates and return mean absolute error

```
def linear_regression_with_degree_mae(X_train,Y_train,X_test,
Y_test,degree):
```

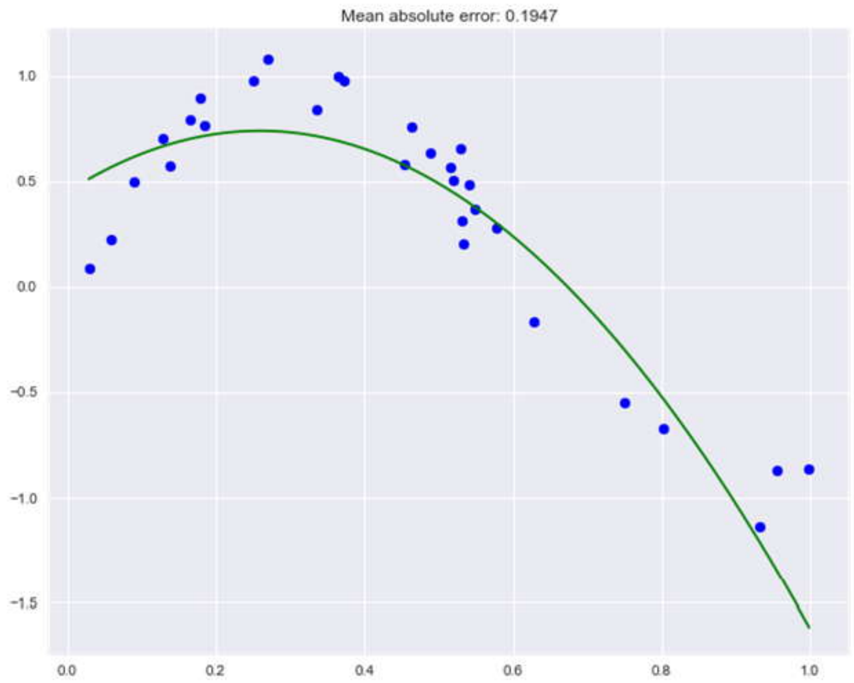
```
    #Make pipeline for higher degree feature
    model = make_pipeline(PolynomialFeatures(degree),
    LinearRegression())
    model.fit(pandas.DataFrame(X_train), pandas.
    DataFrame(Y_train))

    #Change training and test data as requirement
    X_test = pandas.DataFrame(X_test,columns=['X'])
    X_test = X_test.sort_values(by=['X'])
    Y_test = pandas.DataFrame(Y_test)
    Y_test = Y_test.ix[X_test.index]

    #Generate predictions
    Y_pred = model.predict(X_test)
    return mean_absolute_error(Y_test,Y_pred)
```

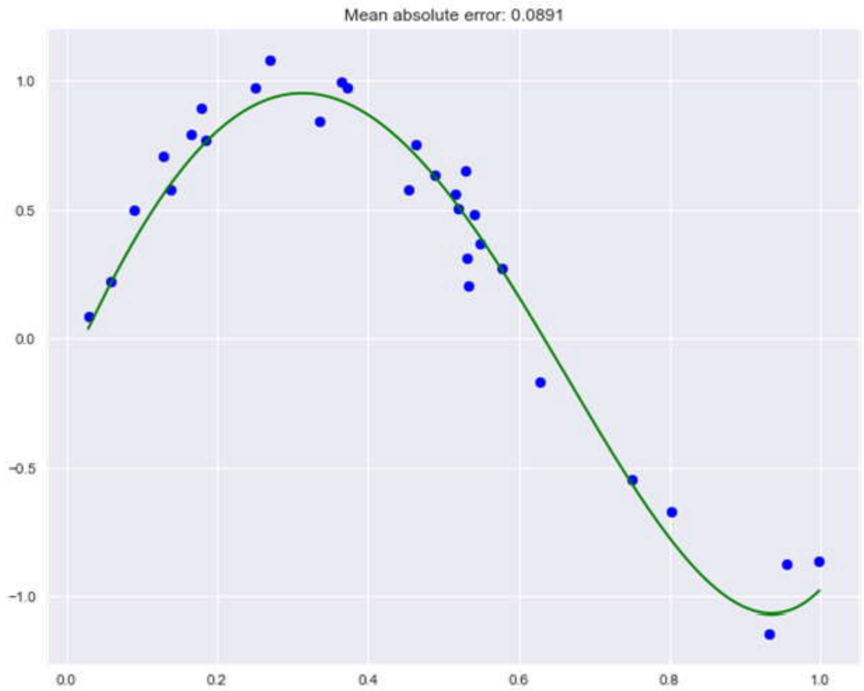
Fit model with degree = 2

```
linear_regression_with_degree(X_train,Y_train,X_test,Y_test,2)
```



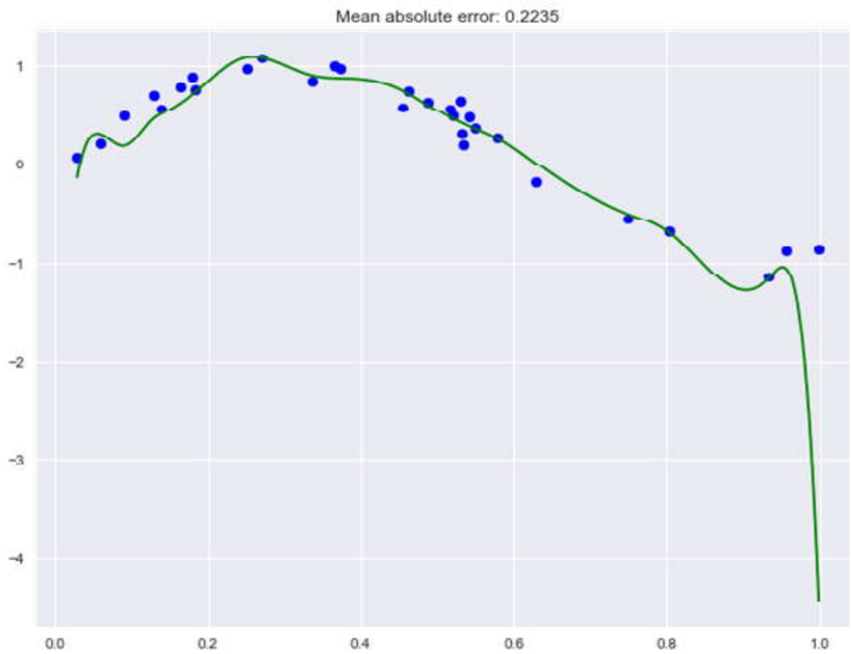
Fit model with degree = 4

```
linear_regression_with_degree(X_train,Y_train,X_test,Y_test,4)
```



Fit model with degree = 16

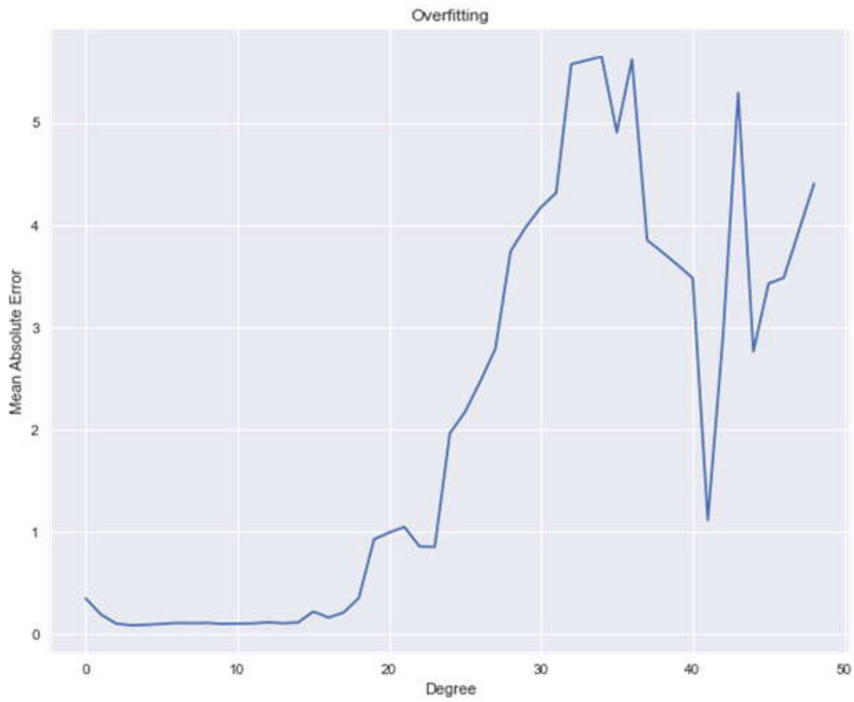
```
linear_regression_with_degree(X_train,Y_train,X_test,Y_test,16)
```



As, we can see that increasing degree (or complexity) will not always help. On test data, degree=4 is giving good result, as compare to degree= 2. When we increased the degree to 16, performance reduces. Model is trying to fit accurately on training data and performing bad on generalized data.

```
error_values = list()
for i in range(1,50):

error_values.append(linear_regression_with_degree_mae(X_train,
Y_train,X_test,Y_test,i))
plt.plot(error_values)
plt.ylabel('Mean Absolute Error')
plt.xlabel('Degree')
plt.title('Overfitting')
```



The problem of overfitting influenced by number of features. If data contains more features, overfitting is much harder to avoid. We can reduce overfitting by increasing the data points.

To reduce overfit, we can change our model assessment. Earlier we are concerned only about fit on the data. Now, we will consider parameters of model along with fit on the data that controls overfitting problem.

So, our new cost for any model would be the sum of measure of fit and magnitude of coefficients of model. Small number in both is desired. So, we need to balance these two numbers to reduce the overall cost.

Measure of fit can be calculated same as earlier we calculated the Residual Sum of Square

$$RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Coefficient magnitude can be handled by different ways. One way is to sum the absolute values of coefficients

$$\|\beta\|_1 = \|\beta_0\| + \|\beta_1\| + \|\beta_2\| + \ldots + \|\beta_d\|$$

This is known as **L1 norm**.

We can also sum the square of magnitude of coefficients

$$\|\beta\|_2^2 = \beta_0^2 + \beta_1^2 + \beta_2^2 + \ldots + \beta_d^2$$

This is known as L2 norm.

6.2 Ridge Regression

Let's consider first L2 norm. Total cost of the model can be given as:

Cost = $RSS = \|\beta\|_2^2$

We can put a tuning parameter

Cost = $RSS = \gamma\|\beta\|_2^2$

γ is a tuning parameter which is used to control the effect of magnitude of coefficient in the cost of the model. In other words, it controls the balance between RSS and L_2 norm.

If the value of tuning parameter γ is zero, it is same as our earlier solution and cost is equal to RSS only. If the value of γ is very high, we are neglecting the RSS and focusing on magnitude of coefficients only.

For low γ

Bias is low, but variance is high.

For high γ

Bias is high, but variance is low.

So, we need to choose ? considering above factor.

This technique of considering magnitude in L2 norm is known as **ridge regression**.

First, we will try ridge regression on our boston housing data.

```
#Split the data
BX_train, BX_test, BY_train, BY_test = train_test_split
boston_data_X, boston_data_Y, test_size=0.20)
```

Create object of ridge regression:

```
ridge_regression = Ridge()
```

Fit the model:

```
ridge_regression.fit(BX_train,BY_train)
```

Make predictions:

```
BY_pred = ridge_regression.predict(BX_test)
```

Mean Absolute Error:

```
print('Mean absolute error: %.2f' % mean_absolute_error
(BY_test,BY_pred))
```

Mean absolute error: 3.91

We can use different combination of parameters to get the better results. In ridge regression, we can tune the following parameters:

- **alpha:** Regularization strength. Tuning parameter which controls
- **normalize:** Boolean. Normalize data or not.
- **max_iter:** maximum number of iteration.
- **solve:** which solver to use {'auto', 'saga', 'lsqr', 'sparse_cg', 'svd', 'cholesky', 'sag'}

Ridge regression modified:

```
ridge_regression =
Ridge(alpha=0.2,normalize=True,max_iter=1000,solver='cholesky')
ridge_regression.fit(BX_train,BY_train)
BY_pred = ridge_regression.predict(BX_test)
print('Mean absolute error: %.2f' % mean_absolute_error
(BY_test,BY_pred))
```

Mean absolute error: 3.89

Now, cost of the ridge regression is:

$$\text{Cost} = \text{RSS} + \gamma \|\beta\|_2^2$$

$$\text{Cost}(\beta) = (y - X\beta)^T (y - X\beta) + \gamma \beta^T \beta$$

Gradient of the solution is given as:

$$\nabla[(y - X\beta)^T (y - X\beta) + \gamma \beta^T \beta]$$

$$= -2X^T(y - X\beta) + 2\gamma\beta$$

Closed form solution is given by taking cost as zero.

$$0 = -2X^T(y - X\beta) + 2\gamma\beta$$

Solving this we get:

$$\beta = (X^TX + \gamma I)^{-1}X^Ty$$

Where I is the identity matrix.

If $\gamma = 0$

$$\beta = (X^TX)^{-1}X^Ty$$

If γ = infinity

$\beta = 0$

If the dimension of data (number of features) is high, this solution is computationally very expensive.

We can use gradient descent algorithm to get the value of parameters.

If we have large amount of data we can use the approach of dividing the dataset into training, validation and test set. First choose a value of γ and train the model. Evaluate the model on validation set. Do this process for different value of γ. Finally, test the model on test set.

If the dataset is small, we cannot use above approach as validation set do not represent complete dataset.

So, we can use all the training data as validation set and consider the average performance. This technique is known as **K-Fold Cross Validation**. First, we divide the dataset into K.



We train the model K times. Each time consider one set out of K for validation and remaining K-1 for training. Do this for each of the K set and take the error in each model. Final error is the average of all the errors on these K validation sets.

When value of K is equal to number of training example, it gives the best approximation of error. But this is computationally expensive. We must train the model for each data point. This is known as **Leave One Out cross validation**. Normally K is chosen as 5, 10 or 20.

To estimate our tuning parameter, we select some value of γ and calculate cross validation error. Repeat this for each possible value of γ. The value for which cross validation error is lowest, is consider the best.

Ridge regression helps in reducing overfitting. We can apply higher penalty.

```
#First create a function that take number of degree, train
and test data and generate regression curve.
def  ridge_regression_with_degree(X_train,Y_train,X_test,
Y_test,degree,alpha):

    #Make pipeline for higher degree feature
    model = make_pipeline(PolynomialFeatures(degree),
    Ridge(alpha=alpha))
    model.fit(pandas.DataFrame(X_train), pandas. DataFrame
    Y_train))

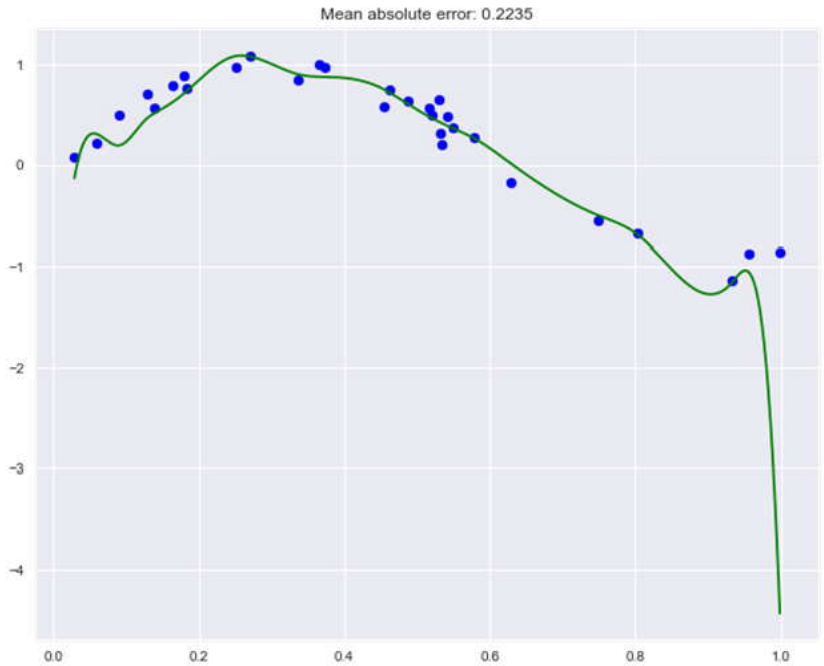
    #Change training and test data as requirement
    X_test = pandas.DataFrame(X_test,columns=['X'])
    X_test = X_test.sort_values(by=['X'])
    Y_test = pandas.DataFrame(Y_test)
    Y_test = Y_test.ix[X_test.index]

    #Generate predictions
    Y_pred = model.predict(X_test)
    #Perform spline smoothing. Higher degree lines will be
    much smooth.
    smooth_feature = numpy.linspace(numpy.min(X_test['X'].
    tolist()), numpy.max(X_test['X'].tolist()),1000)
    smooth_points =
    spline(X_test['X'].tolist(),Y_pred,smooth_feature)

    #Plot the curve
    plt.scatter(X_test, Y_test, color='blue')
    plt.plot(smooth_feature,smooth_points,'-g')
    plt.title('Mean absolute error: %.4f' % mean_absolute_error
    (Y_test,Y_pred) )
```

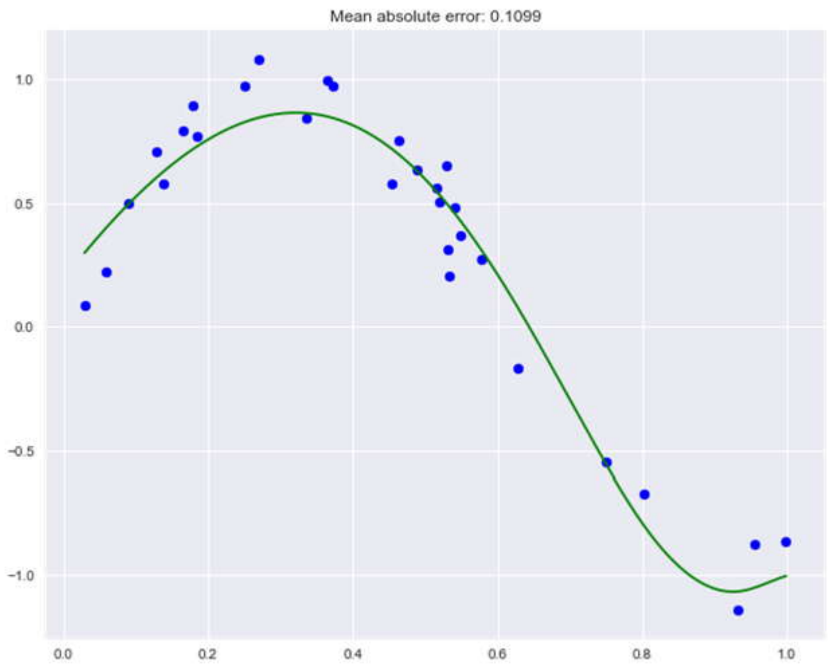
alpha = 0 (Same as linear regression least square solution)

```
ridge_regression_with_degree(X_train,Y_train,X_test,
Y_test,16,0)
```



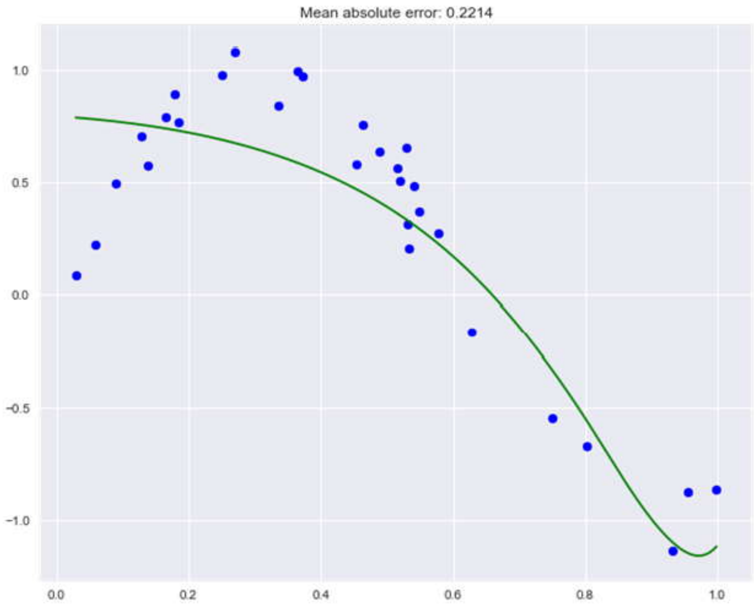
alpha = 0.01

```
ridge_regression_with_degree(X_train,Y_train,X_test,Y_test,16,0.01)
```



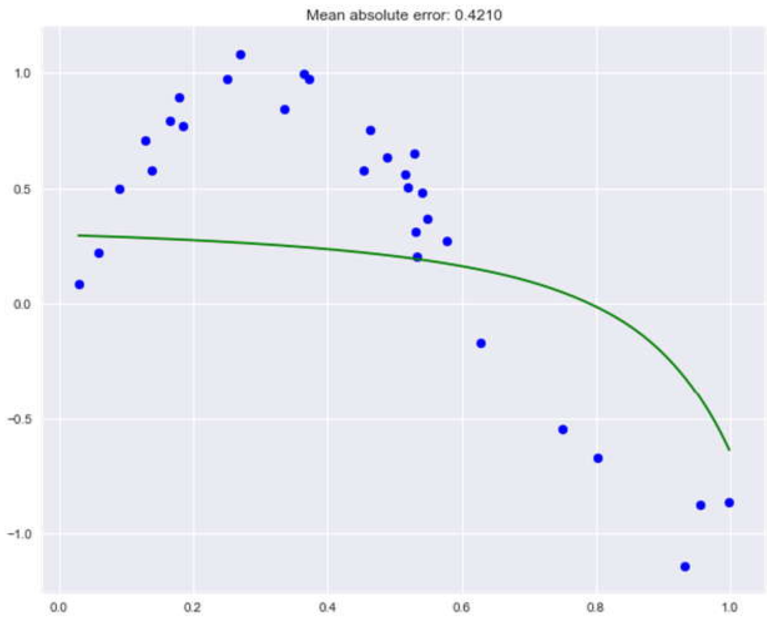
alpha = 1

```
ridge_regression_with_degree(X_train,Y_train,X_test,Y_test,16,1)
```



alpha = 100

```
ridge_regression_with_degree(X_train,Y_train,X_test,Y_test,16,100)
```



6.3 Lasso Regression

Good features are always necessary for any model to perform well. But having lots of features, we need large computation to build the model. Consider a problem where features are words present in dictionary. These features can go to millions. If we consider combination of words, these numbers can go to billions. If we run regression algorithm over it, model needs to find billions of parameters and take huge time in training.

Feature selection is a process of selecting useful features. We tried some techniques in feature engineering chapter. We will use lasso regression to do this work.

6.3.1 All Subset algorithm

One way to do is to find all possible combinations of features and train model for each combination.

For i =1 to number_of_features:

    Take all the combinations of i features

    Train model on selected model

    Calculate Error by using cross validation

Select the model with least error.

This algorithm has run time complexity in exponential. This algorithm is good if number of features are less. But if number of features are large, it is computationally very expensive to apply this algorithm.

If we have total D number of features, we have to train and test 2D models.

Number of features (D)	Number of models
4	16
8	256
16	65536
32	4294967296

Clearly it is not a good technique for large features.

6.3.2 Greedy Algorithm for Feature Selection

Greedy algorithm based on finding the best feature within the available feature set. We add that feature in our list and remove that feature from available list.

Feature set = empty

Available set = All features

For i =1 to number_of_feature:

    Fit the model by using current best feature and feature set.

    Remove selected feature from available list and put it in Feature set.

    Calculate Error using cross validation

Select the feature set with least error.

Training error will go down in each step. We have to use validation set approach to decide the stopping point.

In this algorithm first, we train the model D times. Next time we have to train the model only $D - 1$ times because one feature is moved to feature set. Third time we have only $D - 2$ features to create model. The algorithm ends with only one feature left. Considering that feature means considering all the features ($D - 1$ in the feature set and 1 in the available set, $D - 1 + 1 = D$). So, the complexity of the algorithm would be

$$D + D\text{-}1 + D\text{-}2 \ldots.. 3 + 2 + 1 = O(D^2)$$

Greedy algorithm is computationally much efficient ($D^2 << 2^D$) than all subset algorithm.

Greedy algorithms can be trained in different ways. Another way to start with all feature and remove features one by one.

6.3.3 Regularization for feature selection

We have seen in ridge regression that we use penalty for higher value of coefficient to save model from overfitting. Value of coefficient may be small but not zero. If we can make the coefficient zero that attribute is not considered in the equation of regression. So, we can remove that attribute. In other words, all non-zero coefficient attributes are selected attributes.

One approach of selecting features with ridge regression is to include the features whose coefficients are greater than some threshold. But this may miss some features whose coefficient weight is less but very much important in making a prediction.

Let's try to change the ridge objective to do this task. In ridge regression, cost of the model is given as:

$$\text{Cost} = \text{RSS} = \gamma \|\beta\|_2^2$$

Instead of L_2 norm, if we use L^1 norm:

$$\text{Cost} = \text{RSS} + \gamma \|\beta\|_1$$

Where

$$\|\beta\|_1 = |\beta_0| + |\beta_1| + |\beta_2| \ldots + |\beta_d|$$

This is called **Lasso Regression**.

γ is a tuning parameter which is used to control effect of magnitude of coefficient in the cost of the model.

As we increase the value of tuning parameter γ, coefficient start moving towards zero. For very large value of γ, very few parameters left in the model (All others are eliminated). When the model run for training iteration, once a coefficient hit a zero, it remains at that value.

One good practice is to normalize the feature before applying regression algorithm. Using normalization, all the feature comes in same range and coefficient weight makes more sense in the model.

Let's work on python code example. Apply lasso regression on boston housing data.

```
lasso_regression = Lasso(alpha=0.001,normalize=True,max_iter=
1000)
lasso_regression.fit(BX_train,BY_train)
BY_pred = lasso_regression.predict(BX_test)
print('Mean absolute error: %.2f' % mean_absolute_
error(BY_test,BY_pred))
```

`Mean absolute error: 3.90`

Similar to ridge regression, we can also choose parameters for lasso regression.

```
#First create a function that take number of degree, train
and test data and generate regression curve.
def  lasso_regression_with_degree(X_train,Y_train,X_test,
Y_test,degree,alpha):

    #Make pipeline for higher degree feature
    model = make_pipeline(PolynomialFeatures(degree),
    Lasso(alpha=alpha))
    model.fit(pandas.DataFrame(X_train), pandas.DataFrame
    (Y_train))

    #Change training and test data as requirement
    X_test = pandas.DataFrame(X_test,columns=['X'])
    X_test = X_test.sort_values(by=['X'])
    Y_test = pandas.DataFrame(Y_test)
    Y_test = Y_test.ix[X_test.index]

    #Generate predictions
    Y_pred = model.predict(X_test)
    #Perform spline smoothing. Higher degree lines will be
    much smooth.
    smooth_feature = numpy.linspace(numpy.min(X_test['X'].
    tolist()),numpy.max(X_test['X'].tolist()),1000)
    smooth_points = spline(X_test['X'].tolist(),Y_pred,
    smooth_feature)

    return X_test,Y_test,smooth_feature,smooth_points,
    mean_absolute_error(Y_test,Y_pred)
```

Create subplot

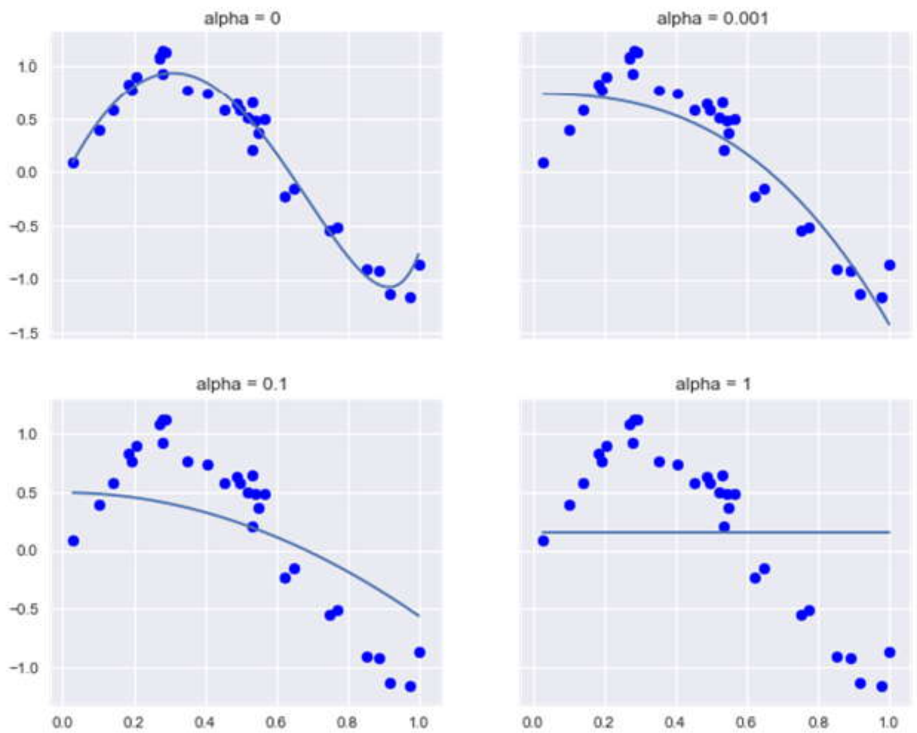
```
f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex='col',
sharey='row')
```

```
A,B,C,D,error =
lasso_regression_with_degree(X_train,Y_train,X_test,Y_test,16,0)
ax1.scatter(A, B, color='blue')
ax1.plot(C,D)
ax1.set_title('alpha = 0 ')

A,B,C,D,error =
lasso_regression_with_degree(X_train,Y_train,X_test,Y_test,16,0.01)
ax2.scatter(A, B, color='blue')
ax2.plot(C,D)
ax2.set_title('alpha = 0.001 ')

A,B,C,D,error =
lasso_regression_with_degree(X_train,Y_train,X_test,Y_test,16,0.1)
ax3.scatter(A, B, color='blue')
ax3.plot(C,D)
ax3.set_title('alpha = 0.1 ')

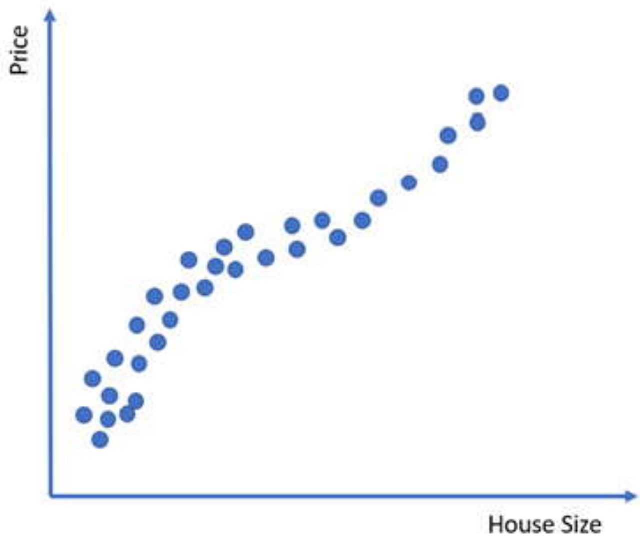
A,B,C,D,error =
lasso_regression_with_degree(X_train,Y_train,X_test,Y_test,16,1)
ax4.scatter(A, B, color='blue')
ax4.plot(C,D)
ax4.set_title('alpha = 1 ')
```



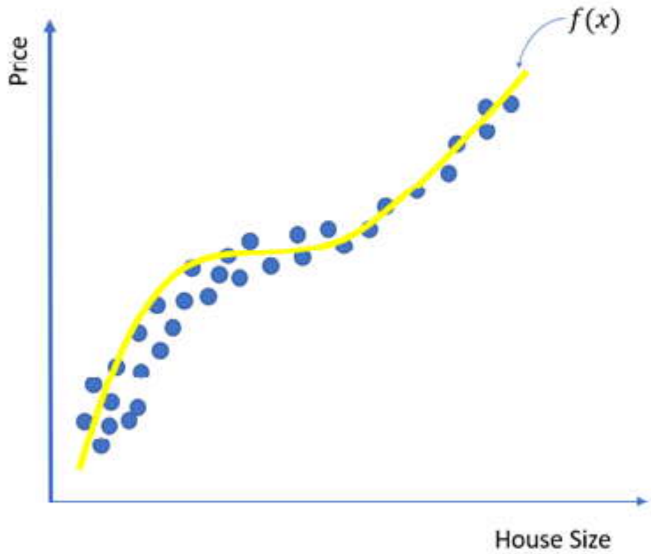
6.4 Non - Parametric Regression

Till now, we are considering only parametric regression techniques. With each type of regression model, parameters are associated. These parameters are global and apply on all kind of data.

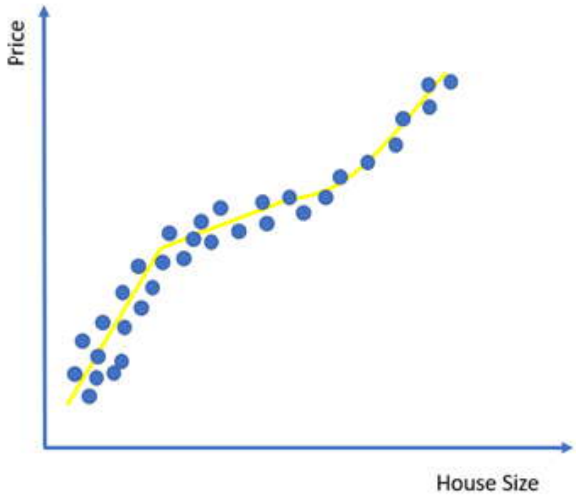
Consider the data points like this:



We can use regression and fit a polynomial like this:



But what if there is no polynomial equation exists to satisfy the points. We can use different equation for different set of points:



In other words, data is not following any global rule or equation. Model can be created by using local structure in data. These approaches are different from any parameter fitting techniques. Complexity can grow as training example increases. We will see two techniques, K-Nearest neighbor regression and Kernel regression in this.

6.4.1 K-Nearest Neighbor Regression

This technique is based on the value of nearest examples in the training set. First consider the value of K, which is 1. This is called **1-Nearest neighbor regression**. The model remembers all the data points in the training set. At the time of prediction, it matches the test example with the nearest training example and output the value of that training example.

Consider the square feet and house price, data is given as:



Blue points are the examples given. We have to predict for Red point. In 1-Nearest neighbor approach we search for nearest point in training example and take that prediction.

If we have only single feature, we can define the distance as Euclidian distance:

$$\text{distance}(x_a, x_b) = |x_a - x_b|$$

But if we have more feature, we can use different ways to calculate the distance.

One way to do that is by using the square root of sum of squared distance in features.

$$\text{distance}(x_a, x_b) = \sqrt{(x_a[1] - x_b[1])^2 + (x_a[2] - x_b[2])^2 + \dots + (x_a[d] - x_b[d])^2}$$

Other distance measures are cosine similarity, rank based distance, correlation based distance etc. We can use different as per the need.

Algorithm for 1NN

DistanceToNN = infinity

Value = 0

For i =1 to number of training records:

    Dist = distance (test example, ith example)

    If(Dist<DistanceToNN) :

        DistanceToNN = Dist

        Value = Value of ith example

Return Value

To give more confidence to our model, instead of looking at one example, we can look at K most similar examples. We can average the prediction of each near example and report the outcome.

Algorithm for K-NN

DistanceToNN = sort(distance from 1st example, distance from Kth example)

Value = index in distance sorted order

For i =1 to number of training records:

    Dist = distance (test example, ith example)

    If (Dist< any example in DistanceToNN):

        Remove the example from DistanceToNN and Value.

        Put new example in DistanceToNN and Value in sorted order.

Return average of Value

Fit using K-NN is more reasonable than 1-NN. K-NN affects very less from noise if dataset is large.

In K-NN algorithm, we can see jump in prediction values due to unit change in input. The reason for this is due to change in neighbors. To handle this situation, we can use weighting of neighbors in algorithm. If the distance from neighbor is

high, we want less effect from that neighbor. If the distance is low, that neighbor should be more effective than others.

$$\hat{y} = \frac{C_{NN1}y_{NN1} + C_{NN2}y_{NN2} + \ldots + C_{NNk}y_{NNk}}{\sum_{p=1}^{k} C_{NNp}}$$

One way to calculate C_{NNp} is:

$$C_{NNp} = \frac{1}{\text{distance}(x_a, x_p)}$$

Where, x_a is the example for which we want prediction and x_p is the value from training example in consideration.

To define the weights of neighbors, we can use kernels.

C_{NNp} = kernel(distance(x_a, x_p))

Kernels are functions which gives more meaningful value to distance.

Code example on boston housing data:

```
knn_regression = KNeighborsRegressor(n_neighbors=3)
knn_regression.fit(BX_train,BY_train)
BY_pred = knn_regression.predict(BX_test)
print('Mean absolute error: %.2f' % mean_absolute_error
(BY_test,BY_pred))
```

Mean absolute error: 4.50

We can try different value of K and check the result.

6.4.2 Kernel Regression

In kernel regression, instead of weighting nearest neighbors, we weight all the training example.

So, prediction is done by using all the training example

$$\hat{y} = \frac{\sum_{p=1}^{n} C_{NNp}y_{NNp}}{\sum_{p=1}^{n} C_{NNp}}$$

$$\hat{y} = \frac{\sum_{p=1}^{n} \text{kernel}(\text{distance}(x_a, x_p))\, y_{NNp}}{\sum_{p=1}^{n} \text{kernel}(\text{distance}(x_a, x_p))}$$

We can also use width of the example by using bandwidth.

Code example on boston housing data:

```
kernel_regression = KernelRidge(alpha=0.5)
kernel_regression.fit(BX_train,BY_train)
BY_pred = kernel_regression.predict(BX_test)
print('Mean absolute error: %.2f' % mean_absolute_error
(BY_test,BY_pred))
```

```
Mean absolute error: 3.73
```

Chapter 7

Classification

Classification is a supervised machine learning technique. In this we have given features and a categorical output. We have to predict this output by using these features. Models are called classifiers. The goal of a classifier is to learn the relationship between input variables and categorical outcome. These different categorical outcomes are called classes.

Let's understand classification using these examples

- **Classifying emails:** We get a lot of emails daily. Some emails contain raw text and fake discounts for products. These are called spam mails. When we login to our account, our email service provider automatically finds spam and remove it from our inbox. So, it classifies each email as spam or not spam. This is a kind of binary classifier because it classifies email in one of the two categories. Today modern email providers classify emails into more categories like primary, social, forum, promotions etc.
- **Classify Document:** When we start reading a document, we get a sense of the category in which document belongs. For example, news in the newspaper. It can belong to sports, politics, education, market etc. These all are categories. We can create a classifier that takes the document as input and classify it into one of the predefine category.
- **Classify Image:** We can build classifier from image. We can give input as different images of animal and give a label to each image i.e. cat, horse, dog, monkey etc. Images contains features, which can be extracted before building any model over it. We can train the classifier. Now this classifier can take input as any animal and classify them in one of the predefined animal categories.
- **Text recognition:** Recognizing text from handwritten data can be done by using classification. We can define each character, digit and symbol as a class. When we get image of any character, we can classify it into one of the character. Using this we can transform complete hand-written document in a computer readable format with high confidence.
- **Stock prediction:** Using the history of stock performance, current news, growth in the sector of that stocks, performance of similar stocks etc., we can build a classifier that advises for any stock, i.e. Buy, Sell or Hold.

In this chapter we will learn core techniques of classification in details. First, we

start with linear classifier, learn how to make equation by using features and predict class using that equation. We will move to learn logistic regression which also gives probability of each class. We will learn gradient descent to get the best model with the available data. With increasing complexity of the model, overfitting may come in the picture. We will learn how to deal with overfitting in classification task.

For non-linear features and relationship, we will focus on decision trees. We will learn how to create best trees, avoid overfitting and present it graphically. We will also learn about best way to handle missing data in these techniques. Apart from accuracy, we will learn other methods exists for model assessment.

We will learn random forest technique to enhance the power of decision tree and reduce overfitting. We will also work on Naïve Bayes algorithm which assume no relation between features of the dataset.

7.1 Linear Classifiers

In linear classifier, output is weighted sum of inputs. To understand the intuition behind linear classifier, let's take an example of an online book store. Our task is to get the top books to display on the website homepage. We have a data related to books and comments on it related to content of that book. We want to promote those books which got good comments from readers. Also, for each new comment we want to classify it as positive feedback or negative feedback.

Example 1: It is great. Love to read it again and again. - Positive

Example 2: Nicely written. Explained core concepts. - Positive

Example 3: Bad way of writing. Poor mistakes. - Negative

For now, assume that, we have generated weights associated with the important words.

Word	Weight
Nice	1.3
Bad	-2
Great	3.5
Mistake	-1.7
terrible	-2.8
Love	2.4
Like	1.6

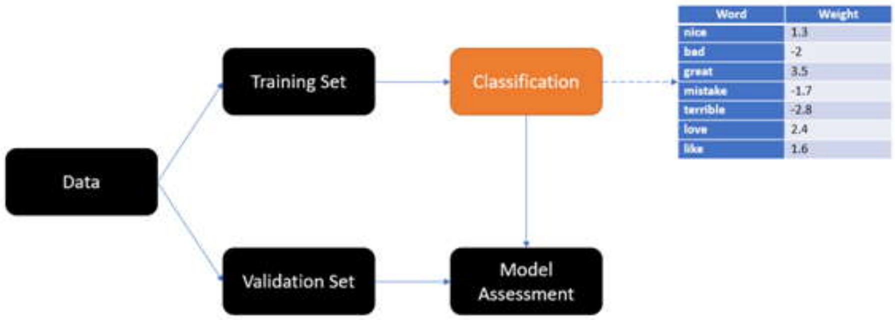
Let's calculate the score associated with the following example:

Test Example: Book is nice. It has some grammatical mistakes but overall it is good read. I like it.

Score(Test-Example): nice + mistakes + like = 1.3 – 1.7 + 1.6 = 1.2 (Positive).

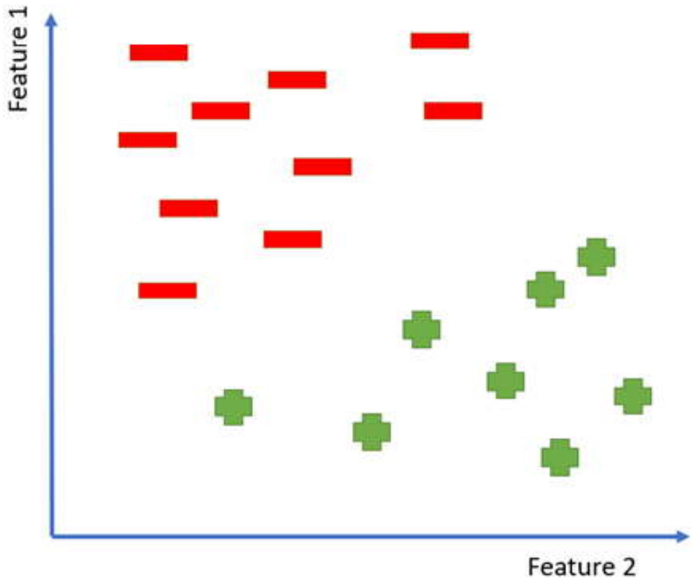
So, the score is greater than zero, we will say that the feedback is positive.

We can understand the procedure from below:

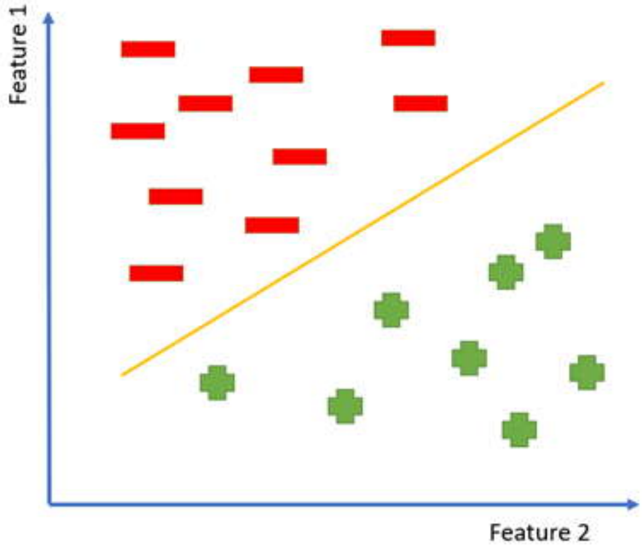


First, we take the data and divide it into training and validation set. On training set we apply classification algorithm and generate classifier model. This model is evaluated on validation set data.

Decision boundaries are boundaries which separate decision of one class from other. Consider a dataset having two numerical features and output is binary (Positive or negative).



Positives are the example of one class and negatives are the example of other class. In this case decision boundary would be a line.



Any point above the line belongs to negative class and any point below the line belongs to positive class.

- If features are 2, decision boundary is a line.
- If features are 3, decision boundary is a plane.
- If features are more than 3 (higher dimensions), decision boundary is a hyperplane.

So, in case of linear classifier our model would be like this:

Model = sign(Score(X))

Score(X) = $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_d x_d$

Where, β_0, β_1, β_2... β_d are the coefficients of input $x_0, x_1, x_2, ..., x_d$. Where, x_0 is taken as 1.

Instead of giving the class, we can give probability of supporting that class. Consider the following statement.

"Book is nice, but I would not recommend others to read it."

For this statement we are not sure about positive or negative feedback. We are not sure about the class. So, instead of giving class, we can give probability of each class that this review fall.

Probability of class is the chance that example belongs to that class. For each class, it varies from 0 to 1. If the probability is zero, chances of having that class is None. If the probability is 1, we are 100% sure that example belongs to that class.

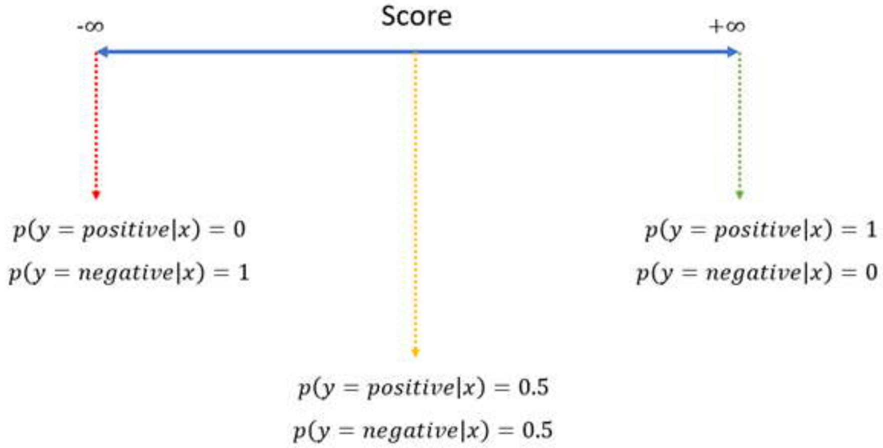
Properties of Probability

- Probability is always between 0 and 1.
- Sum of all the class probability is 1.

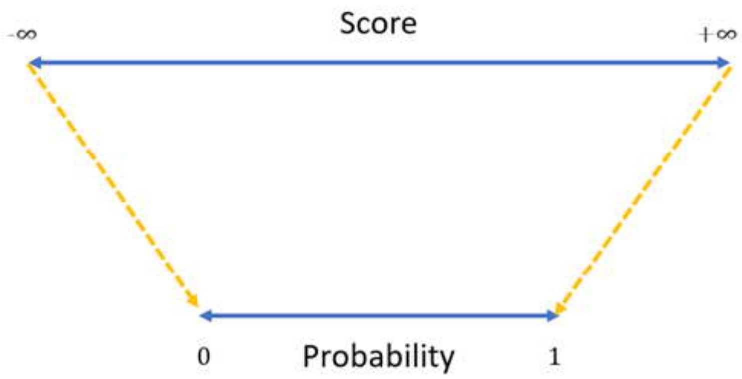
Conditional probability is the probability of an outcome with some conditions are already met. For example, what is the probability that mail is spam given the text of the mail contains the word discount.

We will use this conditional probability to predict class.

Till now we are score for predicting class. If the score is positive, we predict class as positive. If the score is negative, we predict class as negative.



We want to map these scores to probability ranges from 0 to 1.



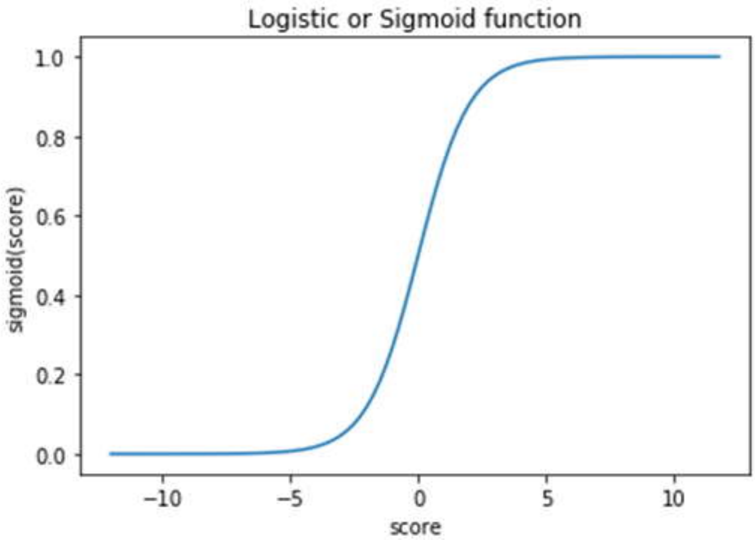
We will do this by using a link function g. The model is same as regression model. After getting the equation, we shrink the value to 0 to 1 range using link function.

7.2 Logistic regression

Logistic regression is a linear model with logistic function as link function. Logistic function is also called as sigmoid function. It is given as:

$$\text{sigmoid(score)} = \frac{1}{(1) + e^{-\text{score}}}$$

This function behaves like this for different values of score:



From above it is clear that, for large negative value, sigmoid returns approximate zero (or very near to zero). For large positive value sigmoid returns nearly one value.

So, probability of class positive can be described as:

$$\text{prob}(y = \text{positive} \mid x, w) = \frac{1}{(1 + e^{-\text{score}})}$$

$$\text{prob}(y = \text{positive} \mid x, w) = \frac{1}{\left(1 + e^{-\beta^T x}\right)}$$

$$\text{prob}(y = \text{positive} \mid x, w) = \frac{1}{\left(1 + e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_d x_d}\right)}$$

Let's take some values of score and check probability of review to be positive:

Score	Prob(y=positive|x,w)
-10	0.0000454
-5	0.00669
-1	0.2689
0	0.5
1	0.7310
5	0.993
10	0.9999

If the values of coefficients are high, curve is steeper.

So, to build a model, we start with a line that fits the data and move the line in a way that maximize the quality or likelihood. We use same gradient descent algorithm which was used in regression to find the parameter of model.

If the input contains categorical input (gender, age group), we need to handle them accordingly. We cannot put these categorical inputs directly in the equation. So, we use encoding and convert them to numerical features. These techniques are already discussed in feature engineering chapter.

Till now we are discussing only binary classification (two classes). If the outcome has multiple classes, we can use the same model to predict one of them.

One approach to deal with multiclass is one-versus-all classification. We build a separate model for each class. Let say we have N number of data points and K classes. We build K different logistic regression model with the data. In each model, we take one class, and all other classes as one class to build the same two class model. For example, if we have to classify an image into three categories, dog, cat, mouse, first we create a model that has one class as dog and second class as other having all examples of cat and mouse. Second model has one class as cat and second class as other having all examples of dog and mouse. Similarly, for third model has one class of mouse and other as dog and cat.

Now, when we get any record to classify in one of the K classes, we predict each class probability by using K different model. The class having maximum probability is reported as predicted class. One important observation is the sum of each class probability is not needed to be 1.

For example, for a test image

By model-1 Prob(dog|x) = 0.4

By model-2 Prob(cat|x) = 0.7

By model-3 Prob(mouse|x) = 0.2

Prediction = Class (Max(Prob(dog|*x*), Prob(cat|*x*), Prob(mouse|*x*))) = cat

Overfitting

In regression we learned about overfitting and how it impacts the performance of the model. Overfitting exists in classification also.

As per the flow, we first start by training the model on training data. We divide the training set into train and validation set. We create the model by using these two sets. Once the model is done, we evaluate it on test data.

So, test data contains instances or records and a target variable which contains classes. When our model predicts correct class by using the record, we call it success. When it gives incorrect classification, we call it failure. So, error can be calculated as:

$$\text{Error} = \frac{\text{No of Failure}}{\text{Total Number of Instance}}$$

We can also define accuracy as:

Accuracy = 1 – Error

$$\text{Accuracy} = 1 - \frac{\text{No of Failure}}{\text{Total Number of Instances}}$$

$$\text{Accuracy} = 1 - \frac{\text{No of Success}}{\text{Total Number of Instances}}$$

Accuracy= (No of Success)/(Total Number of Instances)

When we increase the model complexity, training error reduces but generalized error increases due to overfitting.

Overfitting exists if:

$$\text{Training Error}(\beta^*) > \text{Training Error}(\hat{\beta})$$

$$\text{Generalized Error}(\beta^*) < \text{Generalized Error}(\hat{\beta})$$

Our objective is to reduce the generalization error. Overfitting exists in classification also. If we keep increasing our model complexity it gives correct answer for all the training data but failed on test/generalized data.

If the coefficients increase, overfitting also increases. So, we use L_2 norm to penalize the coefficients.

So, our objective considering coefficient penalization is

Total Quality = measure of fit + measure of coefficient magnitude

= log of likelihood + L_2 norm

$$= l(\beta) - \gamma \|\beta\|_2^2$$

Here, γ is the tuning parameter. If γ is zero it is same as standard maximum likelihood solution. If γ is large solution depends highly on parameters. So, we need to choose γ intelligently. This can be done by using validation set or cross validation methods as earlier.

So, using regularization, we can reduce overfitting. Similarly, overconfident predictions also reduce.

We can learn the coefficients by using gradient ascent.

While not converged:

$$\beta^{t+1} = \beta^{t} + \tau \nabla l(\beta^{t})$$

$$\text{Total Derivative} = \frac{\partial l(\beta)}{\partial \beta_k} - \tau \frac{\partial \|\beta\|_2^2}{\partial \beta}$$

Instead of L_2 norm we can use L_1 norm. This reduces the coefficients to zero. We can remove that variable. It helps in feature selection task.

Total Quality = log of likelihood + L_2 norm

$$= l(\beta) - \gamma \|\beta\|_1$$

Let's understand the concept of logistic regression with an example. First import the libraries needed for this task.

```
import pandas
import numpy
import seaborn
import sklearn

import matplotlib.pyplot as plt
%matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 10, 8

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report,
confusion_matrix, roc_auc_score, accuracy_score
from sklearn import metrics
import scikitplot
```

While using some libraries, jupyter notebook show warnings related to previous version of methods. We can hide those warnings by:

```
#Hide all the warnings in jupyter notebook
import warnings
warnings.filterwarnings('ignore')
```

In this chapter we will build different model and compare them later. Let's store the result (AUC Score) in a dictionary.

```
#Store result from different models
model_result = {}
```

We will use titanic dataset which is available on kaggle. Dataset is downloaded from https://www.kaggle.com/c/titanic. This dataset is about people on titanic. It contains some attribute and status which is survived or not.

```
#Load data in pandas dataframe
titanic_data = pandas.read_csv('./data/titanic_data.csv')
```

Check the head of the data by:

```
titanic_data.head()
```

This is the description available for features.

- **PassendgerID:** Unique ID of each passenger.
- **Survived:** Passenger was survived or not. { 0 = Not Survived , 1 = Survived}
- **Pclass:** Ticket Class. It contains 3 cateogies. {1 = Upper, 2 = Middle, 3 = Lower}
- **Name:** Name of the passenger.
- **Sex:** male or female.
- **Age:** Age of the passenger.
- **SibSp:** Number of siblings or spouses aboard the Titanic.
- **Parch:** Number of parents or children aboard the Titanic.
- **Ticket:** ticket number
- **Fare:** Fare collected from passenger.
- **Cabin:** Cabin number.
- **Embarked:** Port of Embarkation {S = Southampton, Q = Queenstown, C = Cherbourg}

We don't need all the information above. We can keep only useful fields and drop the rest.

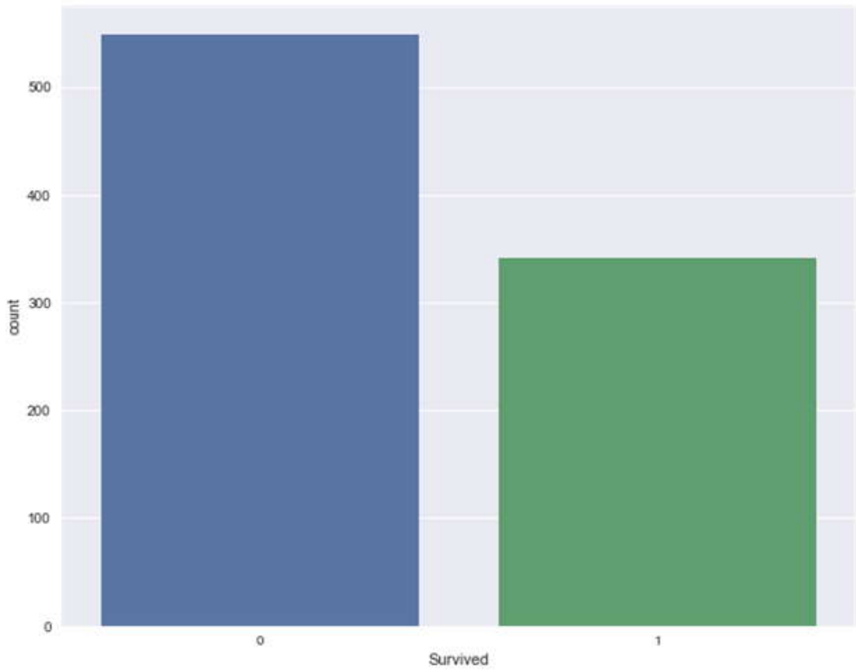
```
#Keep selected columns
titanic_data = titanic_data[['Survived','Pclass','Sex','Age',
'SibSp','Parch','Embarked','Fare']]
```

Check the dimension of the data:

```
titanic_data.shape
```

First, we check the count of survived and not survived passengers. We will use seaborn countplot for this.

```
titanic_data['Survived'].value_counts()
seaborn.countplot(x='Survived',data=titanic_data)
```



Before applying classification algorithm, we will perform feature engineering task on the dataset.

First, we will check for any missing value in the dataset.

```
titanic_data.isnull().any()
```

Now, check the count of null value in each feature.

```
titanic_data.isnull().sum()
```

Age is a continuous variable. It contains 177 null values. We can replace null values by different strategies. Here we are replacing null values by mean value of non-null values.

```
titanic_data['Age'].fillna((titanic_data['Age'].mean()),
inplace=True)
```

Embarked contains two null values. We can drop these two records without impacting much on our algorithm.

```
titanic_data.dropna(inplace=True)
```

Now check the data type of each feature:

```
titanic_data.dtypes
```

Take dependent and independent variables separately

```
titanic_data_X =
titanic_data[['Pclass','Sex','Age','SibSp','Parch','Embarked','Fare']]
titanic_data_Y = titanic_data[['Survived']]
```

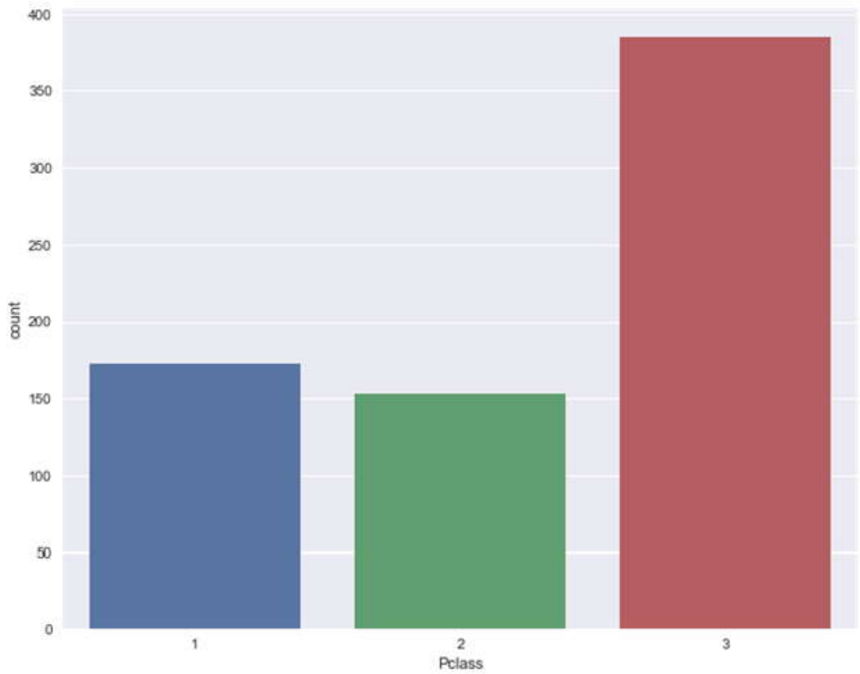
We have to perform more engineering on this dataset. Aim of our classification algorithm is to create a model that works well on unseen data. So, first create a test dataset and keep it separate for testing purpose.

Divide the data into training and test set. train set contains 80% of the data. test set contains 20% of the data.

```
X_train, X_test, Y_train, Y_test = train_test_split
(titanic_data_X, titanic_data_Y, test_size=0.20)
```

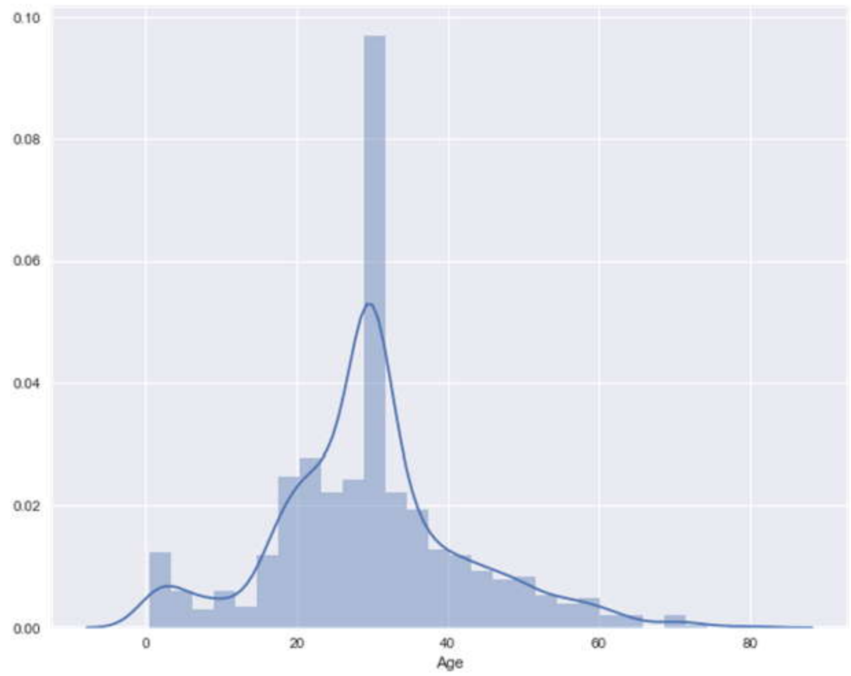
Check the count of Pclass

```
seaborn.countplot(x='Pclass',data=X_train)titanic_data_Y =
titanic_data[['Survived']]
```



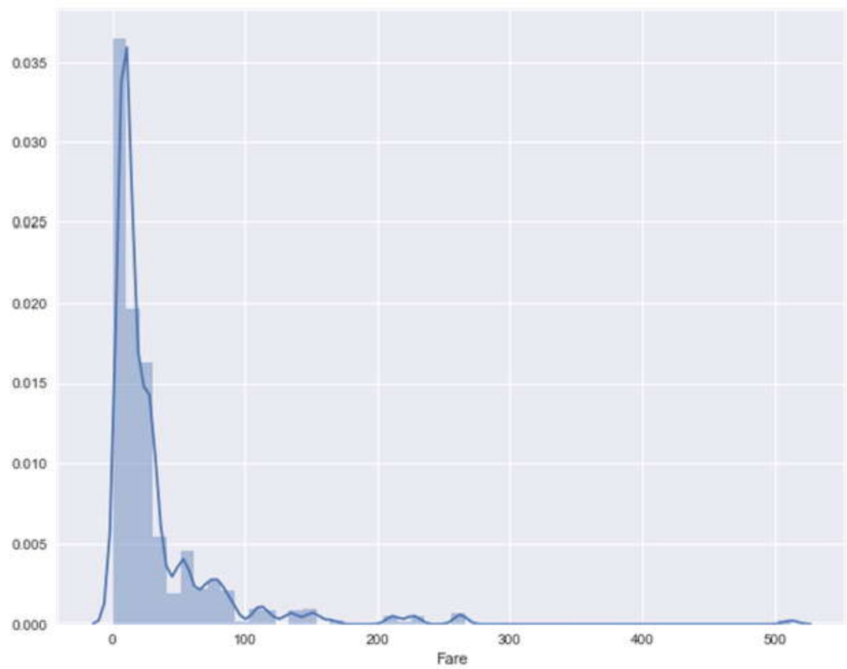
Check the distribution of age data. It contains peak in histogram which is due to replacing of missing value with the mean value.

```
seaborn.distplot(X_train['Age'])
```



Also, distribution of fare.

```
seaborn.distplot(X_train['Fare'])
```



We will perform Z-Score normalization on both these features.

```
age_scaler = StandardScaler()
age_scaler.fit(pandas.DataFrame(X_train['Age']))
```

Transform age.

```
X_train[['Age']] = age_scaler.transform(X_train[['Age']])
```

Perform same normalization with Fare.

```
fare_scaler = StandardScaler()
fare_scaler.fit(pandas.DataFrame(X_train['Fare']))
```

Transform fare.

```
X_train[['Fare']] = fare_scaler.transform(X_train[['Fare']])
```

Change sex feature to 0,1 value.

```
X_train['Sex']  =  X_train['Sex'].map({'female':  0,  'male':
1})
```

Embarked has 3 categories. We can create three different variable which represents each category.

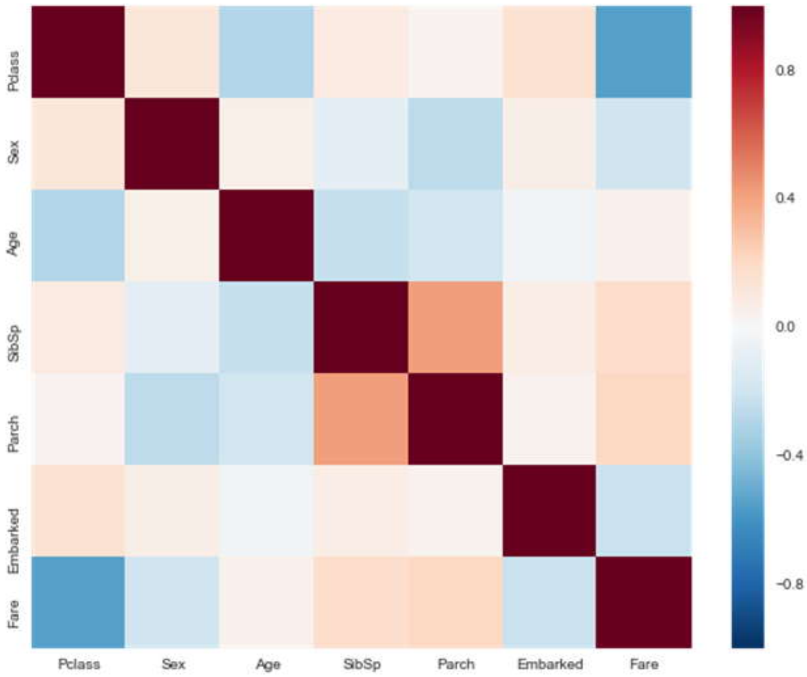
```
embarked_encoder = preprocessing.LabelEncoder()
embarked_encoder.fit(pandas.DataFrame(X_train['Embarked']))
```

Transform Embarked.

```
X_train[['Embarked']] = embarked_encoder.transform(X_train[
['Embarked']])
```

Now, we transform all the features in correct format. Check for correlation between features.

```
seaborn.heatmap(X_train.corr())
```



Fare and Pclass has high correlation. In logistic regression features should not be correlated. So, we can remove one variable.

```
del X_train['Pclass']
```

Now, our data is ready to apply machine learning classification algorithm.

```
X_train_original = X_train
X_train = X_train.values
Y_train = Y_train.values
```

Create object of logistic regression

```
logistic_regression = LogisticRegression()
```

Fit the model

```
logistic_regression.fit(X_train,Y_train)
```

Check the coefficient of model

```
coefficients =
pandas.concat([pandas.DataFrame(X_train_original.columns),
pandas.DataFrame(numpy.transpose(logistic_regression.coef_))],
axis = 1)
coefficients.columns = ['Feature','coefficient']
```

```
coefficients = coefficients.append({'Feature':'Intercept',
'coefficient':logistic_regression.intercept_[0]}, ignore_
index=True)
coefficients
```

Our model is created. We can test it on our test data. But first we need to transform the test data same as we did with training data.

```
def  transform_test_data(test_data,age_scaler,fare_scaler,
embarked_encoder):
    test_data['Sex'] = test_data['Sex'].map({'female': 0,
    'male': 1})
    test_data[['Age']] = age_scaler.transform(test_data
    [['Age']])
    test_data[['Fare']] = fare_scaler.transform(test_data
    [['Fare']])
    test_data[['Embarked']] =
    embarked_encoder.transform(test_data[['Embarked']])
    del test_data['Pclass']
    return test_data
```

Apply function

```
X_test =
transform_test_data(X_test,age_scaler,fare_scaler,embarked_encoder)
```

Take values

```
X_test = X_test.values
Y_test = Y_test.values
```

Predict on test data

```
Y_pred = logistic_regression.predict(X_test)
```

Confusion matrix

```
confusion_matrix(Y_test,Y_pred)
```

Classification report

```
print(classification_report(Y_test, Y_pred))
```

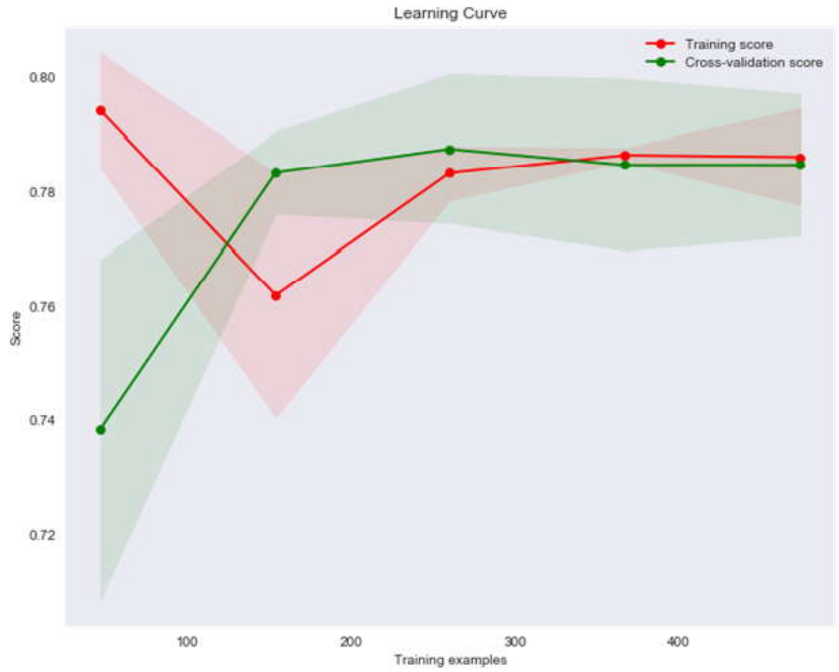
Accuracy

```
print(accuracy_score(Y_test,Y_pred))
```

Now, we will plot different classification result by using Scikit-plot library.

We can plot learning curve of the classifier. It contains two separate curves. One is training curves, and another is cross validation score curve. Initially, Training score is high, but it reduces with increasing number of example. Moving towards generalized solution. CV curve has low value with less data. Its score increases with higher number of training examples

```
scikitplot.estimators.plot_learning_curve(logistic_regression,
X_train,Y_train)
```

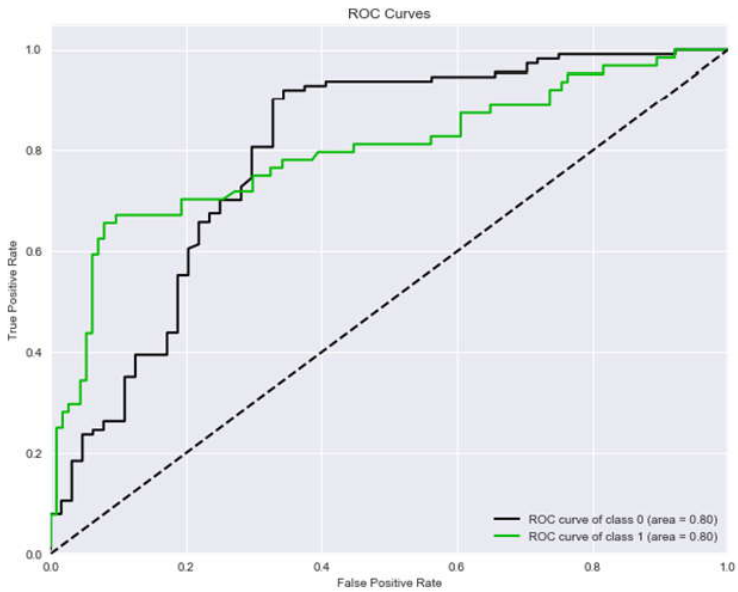


Predict probability instead of class in logistic regression:

```
Y_pred_prob = logistic_regression.predict_proba(X_test)
```

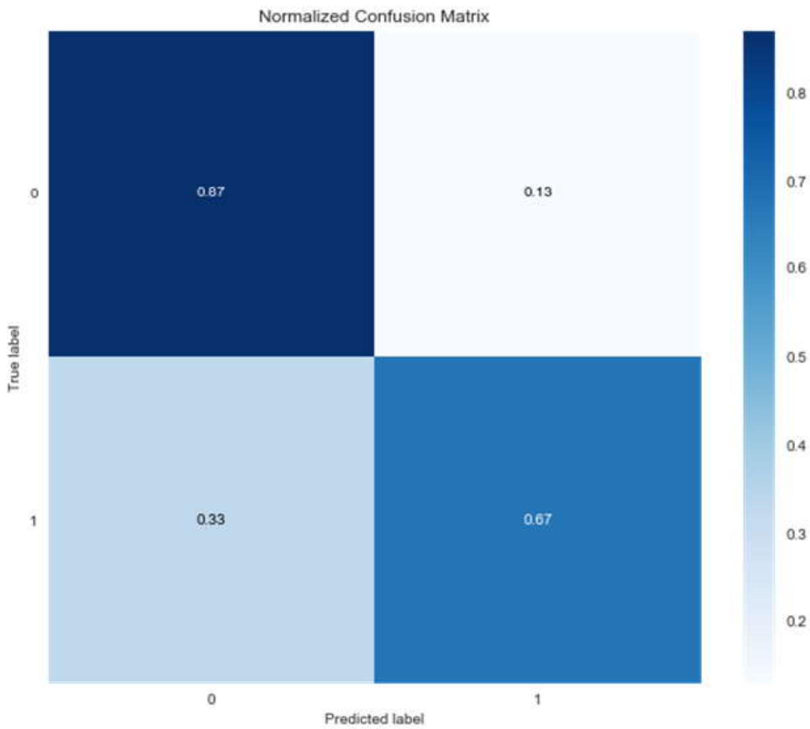
ROC curve:

```
class_1_prob = list()
for i in Y_pred_prob:
    class_1_prob.append(i[1])
print(roc_auc_score(Y_test,class_1_prob))
model_result['Logistic Regression (L2)'] = roc_auc_score
(Y_test,class_1_prob)
scikitplot.metrics.plot_roc_curve(Y_test, Y_pred_prob,
curves=['each_class'])
```



Confusion Matrix:

```
scikitplot.metrics.plot_confusion_matrix(Y_test, Y_pred,
normalize=True)
```



Fit the model with L1 norm:

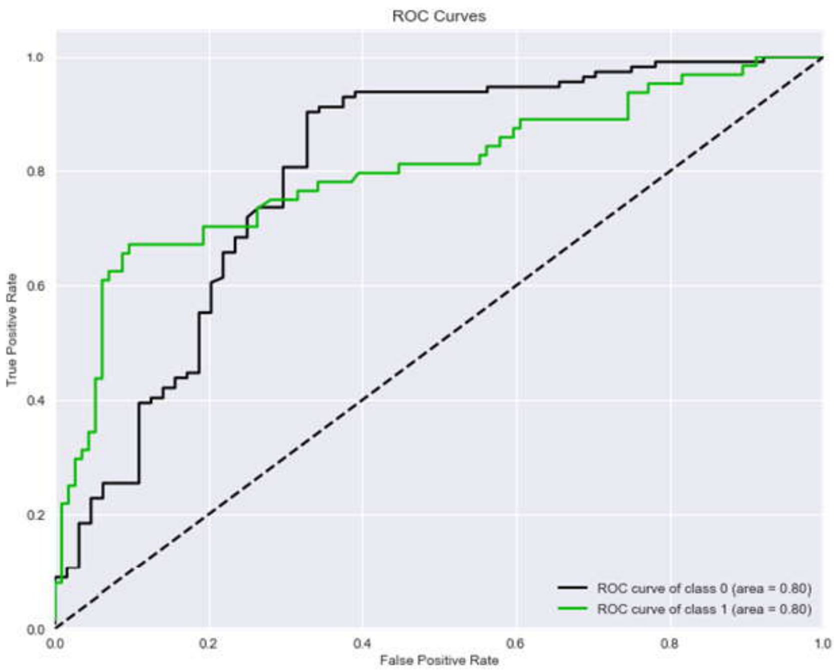
```
logistic_regression_l1 =
LogisticRegression(penalty='l1',class_weight='balanced')
logistic_regression_l1.fit(X_train,Y_train)
```

Check on test data:

```
Y_pred = logistic_regression_l1.predict(X_test)
print(accuracy_score(Y_test,Y_pred))
```

ROC Curve:

```
Y_pred_prob = logistic_regression_l1.predict_proba(X_test)
class_1_prob = list()
for i in Y_pred_prob:
    class_1_prob.append(i[1])
print(roc_auc_score(Y_test,class_1_prob))
model_result['Logistic Regression (L1)'] = roc_auc_score
Y_test,class_1_prob)
scikitplot.metrics.plot_roc_curve(Y_test, Y_pred_prob,curves
=['each_class'])
```

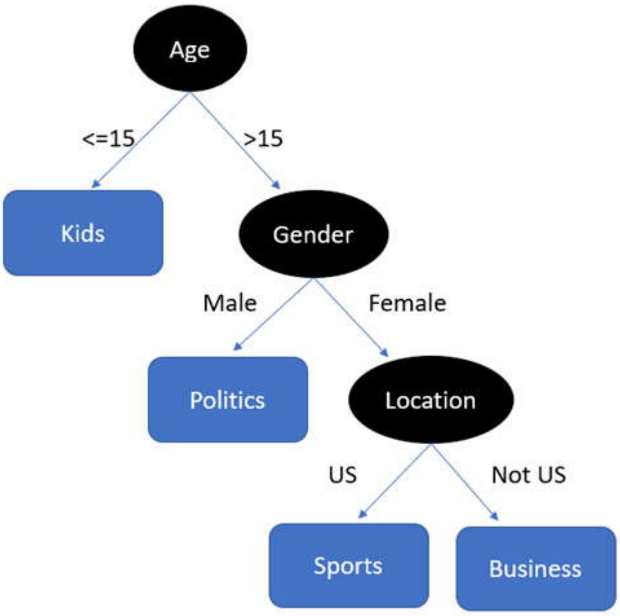


7.3 Decision Trees

Decision tree is a technique of classification. Decision tree is basically a tree in

which each intermediate node represents a decision and leaf node represents a decision. We have learned example of decision tree in our introductory chapter.

Here, is the example of decision tree to classify an article into predefined category.



If the age of a person is equal or less than 15 years, we classify the article as Kids category. If the age is greater than 15 and Gender is Male, article will be classified as Politics.

Advantages of Decision Tree

1. Easy to explain to others. It does not need any complex mathematical knowledge to understand the result.
2. Can handle irrelevant attributes.
3. Can capture nonlinear relationships in the data.
4. Decision tree learning, or construction of tree is fast process. It uses greedy algorithms to create trees. Also, prediction by using tree is fast.
5. Normalization and other cleaning on data is not needed.
6. It is not based on assumptions. In linear methods, we have to test some assumptions before building the model. No assumptions are needed in decision tree learning.
7. Can handle both numerical and categorical attributes.

Disadvantage of Decision Tree

1. Decision tree needs to stop somewhere. Otherwise they will give no error on training data and high error on test data.

Decision trees are mainly of two types:

- **Classification Tree:** Dependent variable is categorical. For example, predict loan defaulter or not using bank statements, age and gender. In classification trees, at any node majority class is given as prediction.
- **Regression Tree:** Dependent variable is continuous. For example, predict salary-based age, years of experience, house location etc. In regression tree, at any node mean value of all the values is given as predictions.

7.3.1 Tree terminology

- **Root Node:** The node at the top of the tree.
- **Child Node:** One node is child node of other if it is below from the node (away from root node).
- **Parent Node:** One node is parent node of other if it is above of the node (closer to the root node).
- **Siblings:** Two nodes are sibling if they share same parent node.
- **Leaf Node:** Node which do not have any child node.
- **Edge:** Connection between two nodes.
- **Degree:** Count of sub tree of any node.
- **Height of a node:** Number of edges from node to leaf node having longest edges in the middle.
- **Height of tree:** Height of root node.

7.3.2 Decision Tree Learning

Now we will focus on algorithm which is used to learn the decision tree from training data. Tree learning algorithms generally works from top to bottom. It checks for best attribute at any point of learning and create a decision node around that. Different algorithms have different definition of best attributes.

Gini Score: It is based on the split of different class attribute at any node. Its best value is zero when node divides the data perfectly (One class completely on one branch and Second class completely on another branch in case of binary classification problem). Its worst value is 1 when both branches have classes in the same proportion. It considers only binary splits while creating tree.

Our goal of decision tree learning is to create a model that reduces error. Error is defined as number of incorrect predictions, divided by total number of examples.

Let's take an example of 3 features and one target variable:

Age Range	Previous Defaulter	Income Range	Loan Status
25-30	No	25000-50000	Approved
30-35	Yes	50000-100000	Approved
25-30	Yes	10000-25000	Not-Approved
20-25	No	10000-25000	Approved
25-30	Yes	25000-50000	Approved

Above data contains 3 features (Age range, Previous defaulters, Income range) and one target variable (Loan status).

We can create a decision tree on each field. Number of possible trees can go to exponential. This is **NP-Hard problem**.

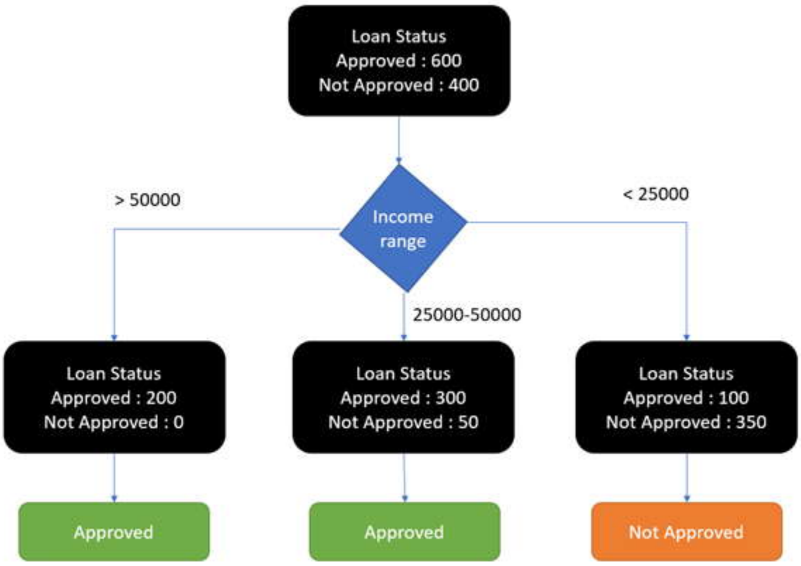
We use recursive algorithm to build decision tree.

- Start with empty decision tree.
- Select best feature to split.
- Split the data into different groups based on feature value.
- If no more possibility of split.
- Make prediction with majority class.
- Split the features recursively.

This algorithm has a lot of open problems to deal with. One is how to select the best feature on which data is going to split. Another is the stopping criterial of recursion. We will solve this issues in this section.

Let's first start with one level decision tree, decision stump. Decision stump contains one decision and classify the data according to that decision.

Let's take an example of Loan status decision tree made on above data. We have past data of 1000 records with 600 as Approved and 400 as Not-Approved.



Above is the decision stump created on income range. It forms three groups from data based on income range. First group contains all approved status. Second group contains 300 approved and 50 Not Approved status. Third group contains 100 Approved and 350 Not Approved cases. We make the decision based on majority of class in that group.

Now looking at the full dataset, let say we have K number of features. We have to decide out of these K feature which feature to split first, second and so on.

So, we start by each feature, create a decision tree and calculate the overall error on each decision stump. The stump which gives lowest error is consider best to split on the data. After splitting data by that decision stump, we move to lower level and build a new decision stump by considering remaining K-1 features. This process moves till all features are finished or nothing remain more to classify.

To make the prediction by using decision tree, we start from the root node and move towards leaf node.

We can also perform multiclass classification with probability. Probability of a class is the number of example of that class in the group divided by total number of examples in the group.

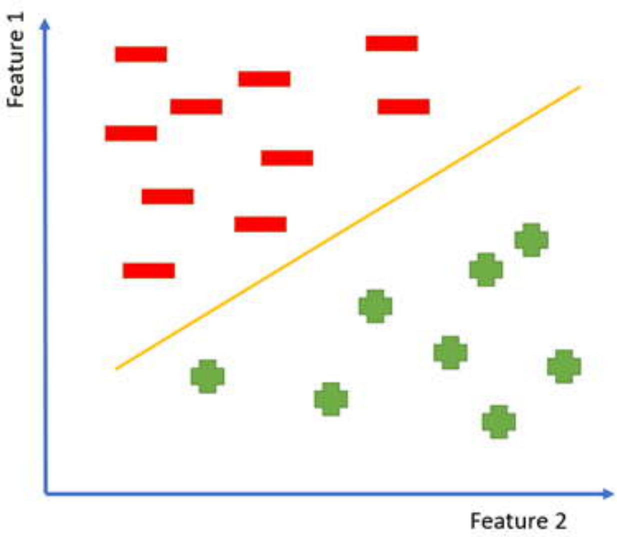
Continuous Variable

Till now we have considered only categorical feature in building decision tree. We can also use continuous feature to build decision tree. One solution is to bucket the continuous data into different groups. For example, divide the income into different income range.

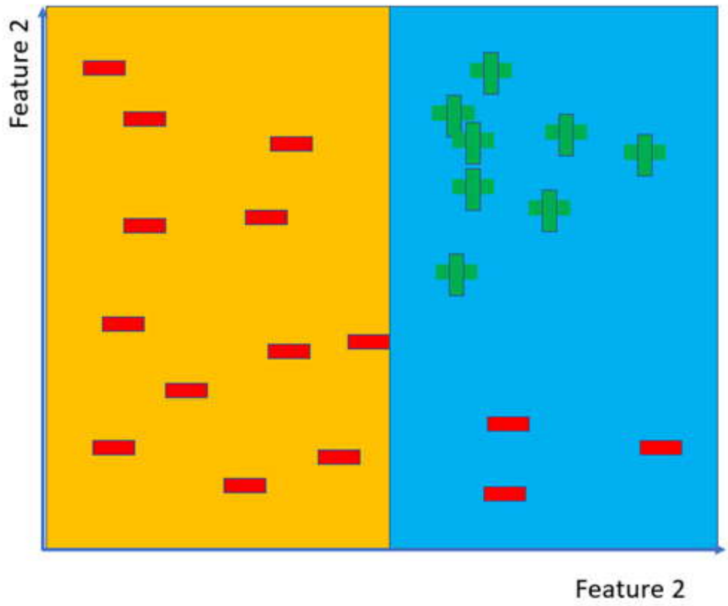
Another solution is to use each possible split. First sort the continuous value in ascending order. Now put a flag or point in between each two points. So, let say if we have 100 data points, we have to make total 99 splits. We create a decision stump for each split and keep the one which gives best split (lowest error).

7.3.3 Decision Boundaries

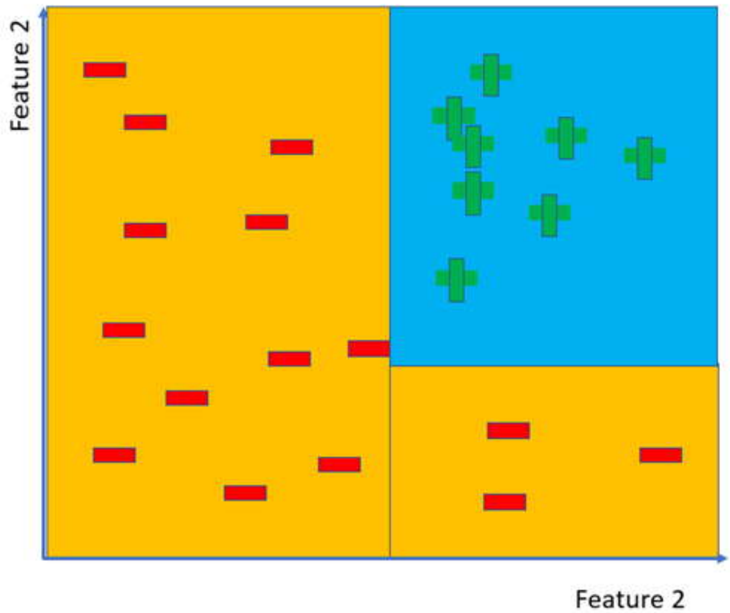
We have seen decision boundaries in logistic regression. Examples explain above the line are classified as negative and below the line are classified as positive.



Now, consider a decision tree. First, we have created a decision tree by using only one features.

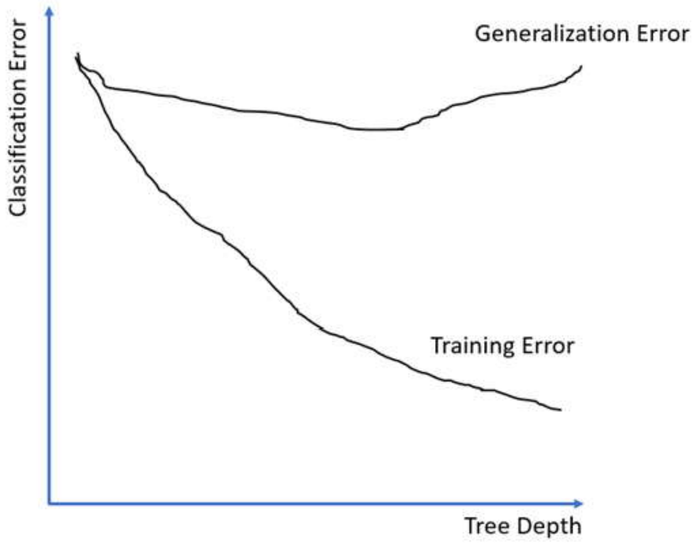


In above, split is made based on the value of feature 2. If we consider split by using both features, decision boundaries can be given as:



So, in decision tree, decision boundaries are always parallel to axis.

Decision tree also faces the problem of overfitting. If we have so many features, we keep splitting and making branches, and end of very less data in each group. This is not the generalized solution. Creating big tree make different small boxes parallel to axis but they are failed on generalized data. Training error keeps decreasing as the depth of the tree increases, but generalization error increases. We can understand the relationship using:



Simple decision trees are better in terms of overfitting. We choose the depth or complexity of decision tree based on the validation tree. If simple and complex tree both gives same validation error (or nearly same), choose the simplest one.

There exists two ways to choose simple tree:

- **Early Stopping:** We start building the tree and stop at some depth (avoid large depth). We can stop where validation error is not decreasing anymore. We can also stop if the data at any node is small. For example, we have 5 data point at one node, one with Yes and 4 as No. We don't want to split again. We can make the prediction with majority class.
- **Pruning:** In pruning, we build the tree first and start cutting the tree to get the better generalization error.

Now, let's build decision tree on the same data. We will build the model and compare the performance.

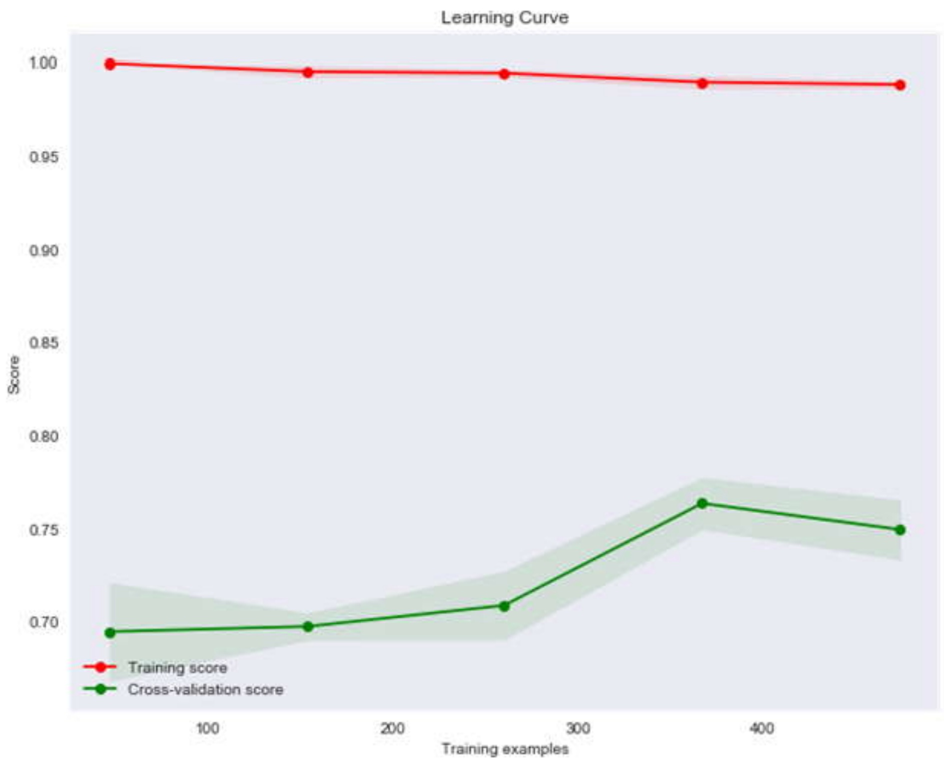
```
decision_tree = tree.DecisionTreeClassifier()
decision_tree.fit(X_train,Y_train)
```

Check on test data:

```
Y_pred = decision_tree.predict(X_test)
Y_pred_prob = decision_tree.predict_proba(X_test)
print(accuracy_score(Y_test,Y_pred))
```

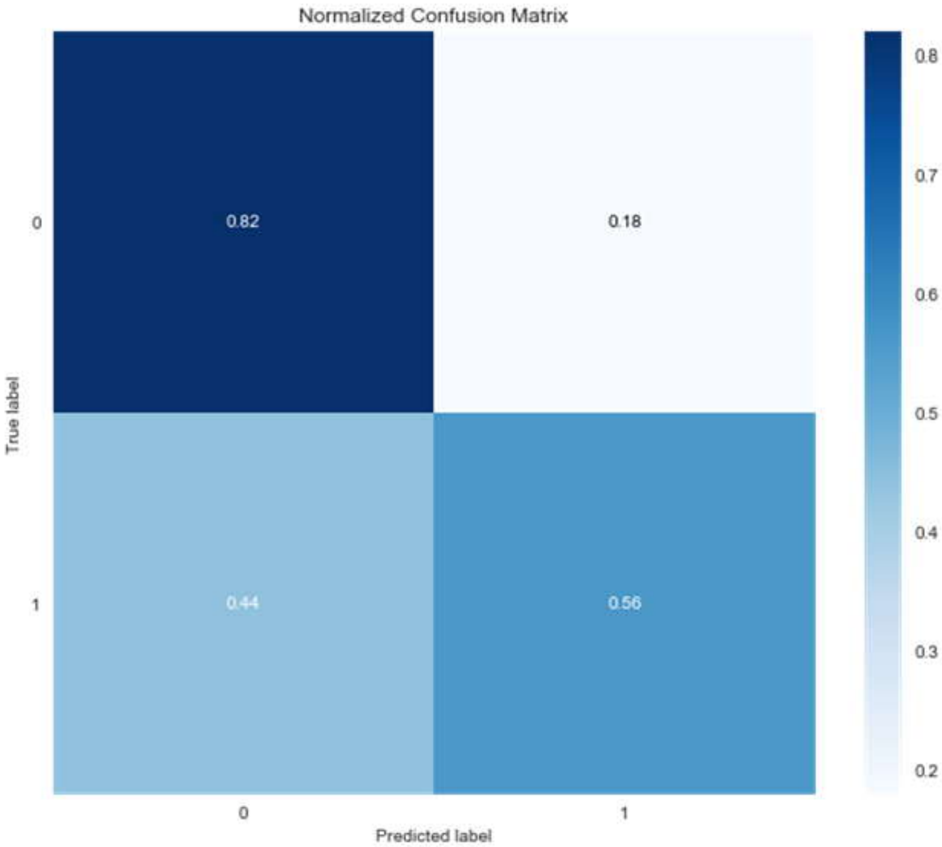
Learning Curve:

```
scikitplot.estimators.plot_learning_curve(decision_tree,
X_train,Y_train)
```



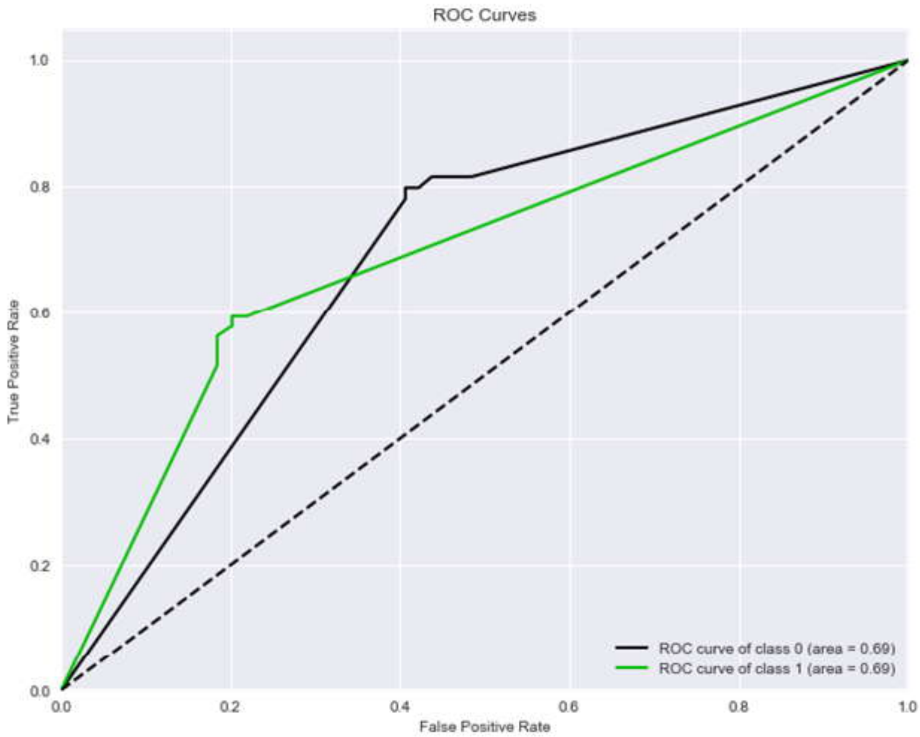
Confusion matrix:

```
scikitplot.metrics.plot_confusion_matrix(Y_test, Y_pred,
normalize=True)
```



Plot ROC Curve:

```
Y_pred_prob = decision_tree.predict_proba(X_test)
class_1_prob = list()
for i in Y_pred_prob:
    class_1_prob.append(i[1])
print(roc_auc_score(Y_test,class_1_prob))
model_result['Decision Tree (Default)'] = roc_auc_score
(Y_test,class_1_prob)
scikitplot.metrics.plot_roc_curve(Y_test, Y_pred_prob,
curves=['each_class'])
```

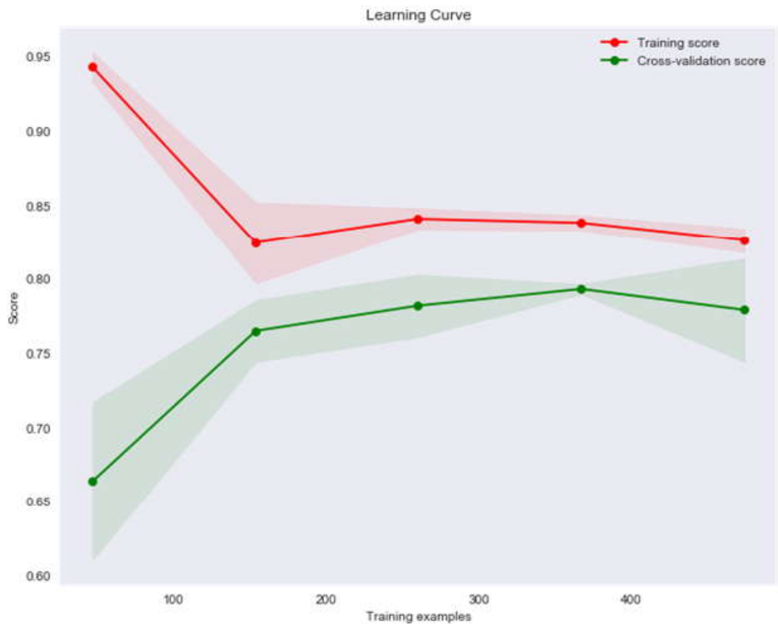


Decision tree improves over accuracy as compare to logistic regression. When we analyze the learning curve, we can see that cross-validation accuracy is not going up, but training error remains high. It is because we are dealing with overfitting. One simple method is to restrict the depth of the tree. Here we choose max depth as 4.

```
decision_tree_4 = tree.DecisionTreeClassifier(max_depth=4)
decision_tree_4.fit(X_train,Y_train)
#Check on test data
Y_pred = decision_tree_4.predict(X_test)
Y_pred_prob = decision_tree_4.predict_proba(X_test)
print(accuracy_score(Y_test,Y_pred))
```

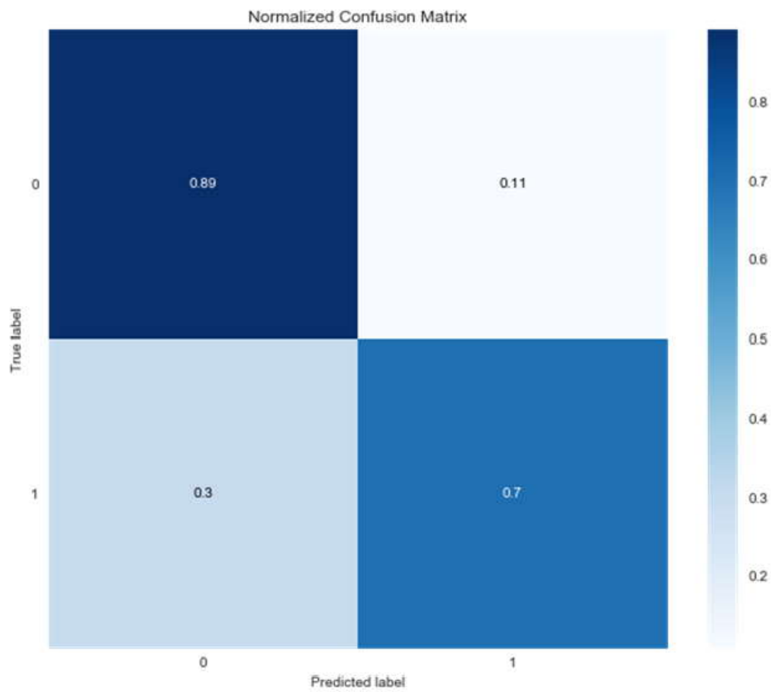
Learning Curve

```
scikitplot.estimators.plot_learning_curve(decision_tree_4,
X_train,Y_train)
```



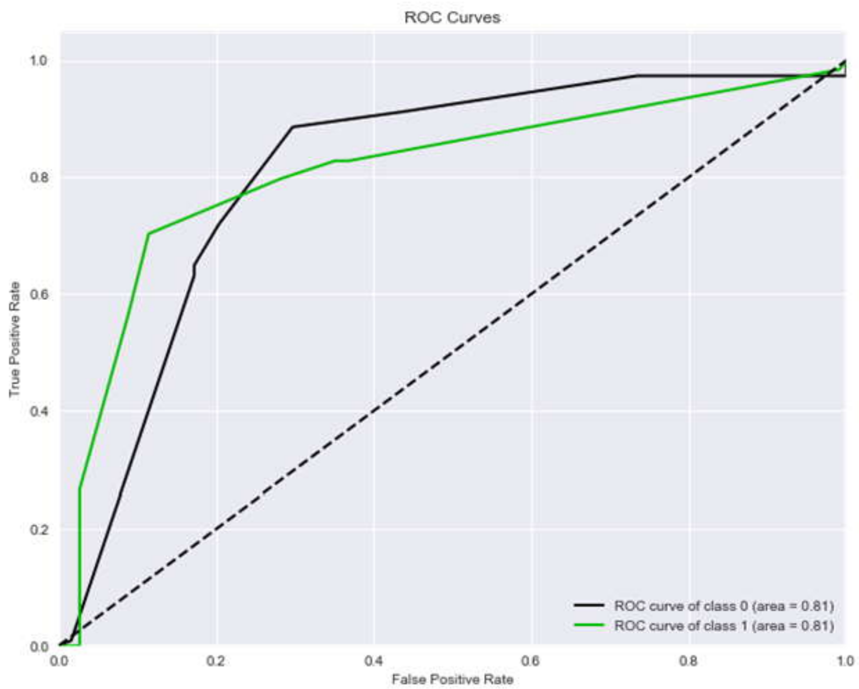
This is our desirable result. Let's check confusion matrix also.

```
scikitplot.metrics.plot_confusion_matrix(Y_test,  Y_pred,
normalize=True)
```



ROC Curve:

```
Y_pred_prob = decision_tree_4.predict_proba(X_test)
class_1_prob = list()
for i in Y_pred_prob:
    class_1_prob.append(i[1])
print(roc_auc_score(Y_test,class_1_prob))
model_result['Decision Tree (Max Depth = 4)'] = roc_auc_score
(Y_test,class_1_prob)
scikitplot.metrics.plot_roc_curve(Y_test, Y_pred_prob,curves=
['each_class'])
```

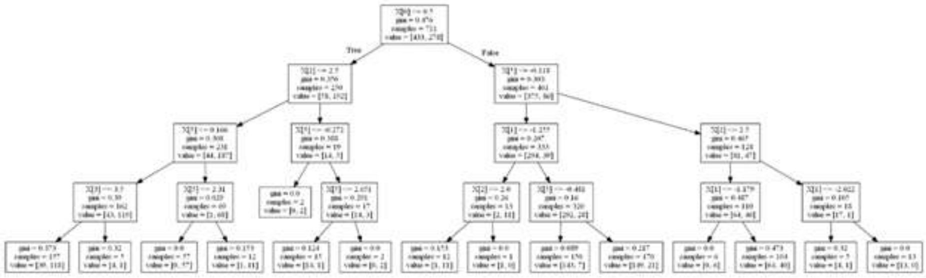


We can also plot the tree and visualize it. Plotting can be done using graphviz: import graphviz

```
dot_data = tree.export_graphviz(decision_tree_4, out_file=
None,
    feature_names=X_train_original.columns,
    class_names=['0','1'],
    filled=True, rounded=True,
    special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Save the tree as png file:

```
tree.export_graphviz(decision_tree,out_file='tree.dot')
!dot tree.dot -Tpng -o tree.png
```



7.4 Random Forest

Random forest is one of the widely used technique in machine learning. This can be used as both classification and regression techniques. In online machine learning competitions this algorithm is used broadly by winning teams.

In decision tree, we used greedy approach to create a single decision tree with the data. We train algorithm with cross validation techniques to avoid overfitting. In decision tree, model is given by a single tree. Prediction is done by traversing the tree from top to bottom. Decisions are taken at the leaf nodes. Random forest, as the name suggest is forest of trees. It contains large number of decision trees which helps in taking a decision. Each tree in random forest is made by same strategy of making single decision tree. While taking a decision, we take vote for all small decision tree and decide the class by majority votes. For example, in a binary classification problem, with different setting, we can create hundreds of decision trees. For prediction, if 80% trees are saying class A (moving from leaf node from root) and 20% trees are saying class B, decision would be class A.

Using random forest instead of decision trees are advantages. Random forest can handle missing data. They can be used for both supervised learning tasks. They can handle large dimensionality in the data. Also using multiple trees instead of single reduces overfitting possibility.

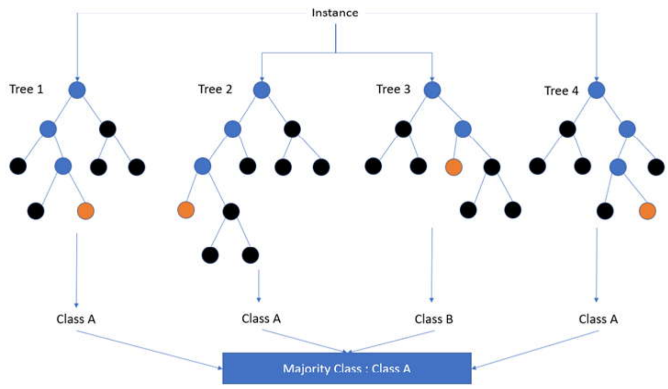
It also has some disadvantages. We have very little control over the model. As the model is the combination of trees, our model is much complex compare to single decision tree. It is difficult to explain because of hundreds or thousands of trees.

Random Forest Algorithm

1. Consider training dataset with N records. Create samples by taking N records from this dataset with replacement. It means first we select a record from N records randomly. Now after choosing it, we again choose a record from N records. In this strategy, record may repeat.

2. Select a number m which is less than number of attributes K. These are the number of attributes considered randomly from total attributes at the time of building small tree (m<K).
3. Build p number of trees with different samples and randomly chosen m attributes.
4. Grow each tree without pruning consider all attributes or where no more possibility of division (Only one class left).
5. Make predictions by using majority voting of all trees.

Random forests are also known as **Ensemble technique**. These techniques are divide and conquer approach. It uses small number of weak learner to generate a strong learner. In random forest, small learners are small trees. Together with the power of majority voting they make strong learner.



Consider the example above. Here we have created four different trees in a random forest model. Circle represents nodes in the tree. Blue circle represent path from root and all the decision nodes comes in the path. Orange node represent the decision taken by the tree. Finally, all the votes are combined, and majority class is reported as the output of complete random forest model.

Now we will create a random forest model on titanic dataset used earlier and evaluate performance.

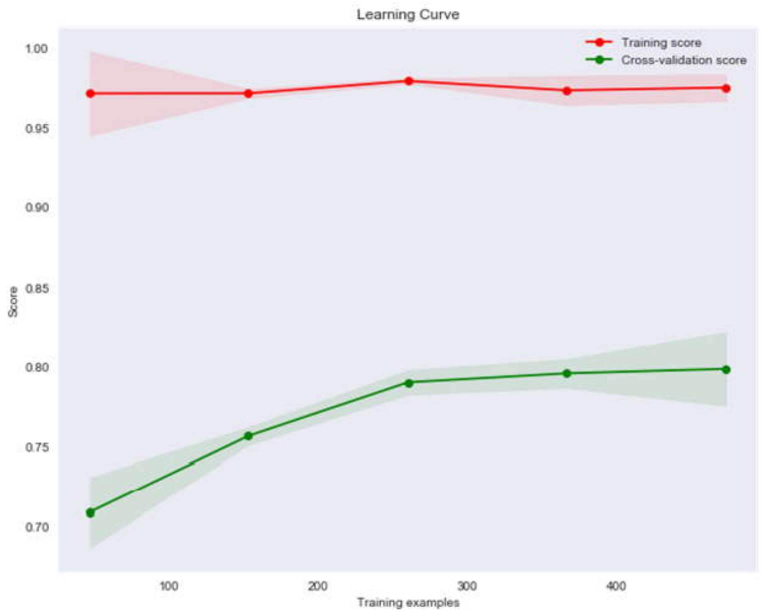
```
random_forest = RandomForestClassifier()
random_forest.fit(X_train,Y_train)
```

Check on test data:

```
Y_pred = random_forest.predict(X_test)
Y_pred_prob = random_forest.predict_proba(X_test)
print(accuracy_score(Y_test,Y_pred))
```

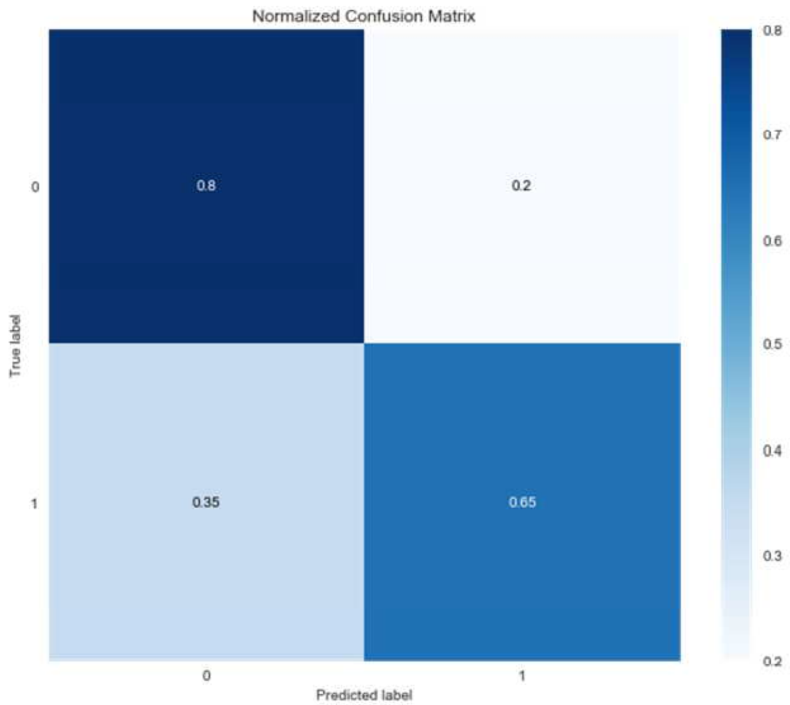
Learning Curve:

```
scikitplot.estimators.plot_learning_curve(random_forest,
X_train,Y_train)
```



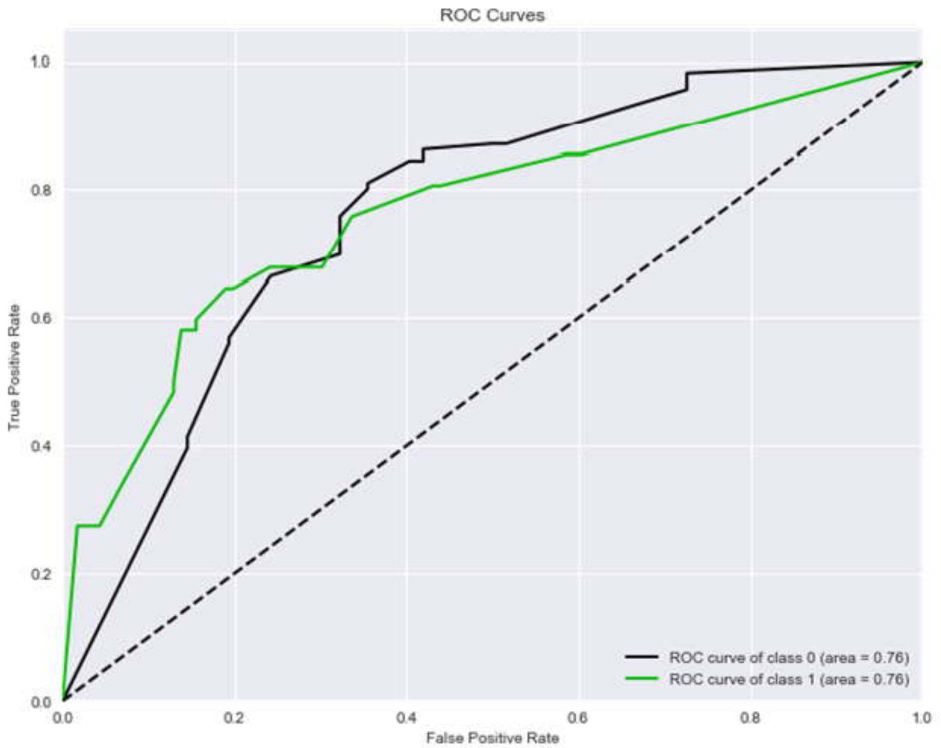
Learning Curve:

```
scikitplot.metrics.plot_confusion_matrix(Y_test, Y_pred,
normalize=True)
```



ROC Curve:

```
Y_pred_prob = random_forest.predict_proba(X_test)
class_1_prob = list()
for i in Y_pred_prob:
    class_1_prob.append(i[1])
print(roc_auc_score(Y_test,class_1_prob))
model_result['Random Forest(Default)'] = roc_auc_score
(Y_test,class_1_prob)
scikitplot.metrics.plot_roc_curve(Y_test, Y_pred_prob,
curves=['each_class'])
```



We used default parameters of random forest. We can change it and retrain the model. We will do three changes in next model building.

max_depth = 5. Each tree can go to depth 5.

n_estimators = 100. Total 100 trees are build and average prediction is taken.

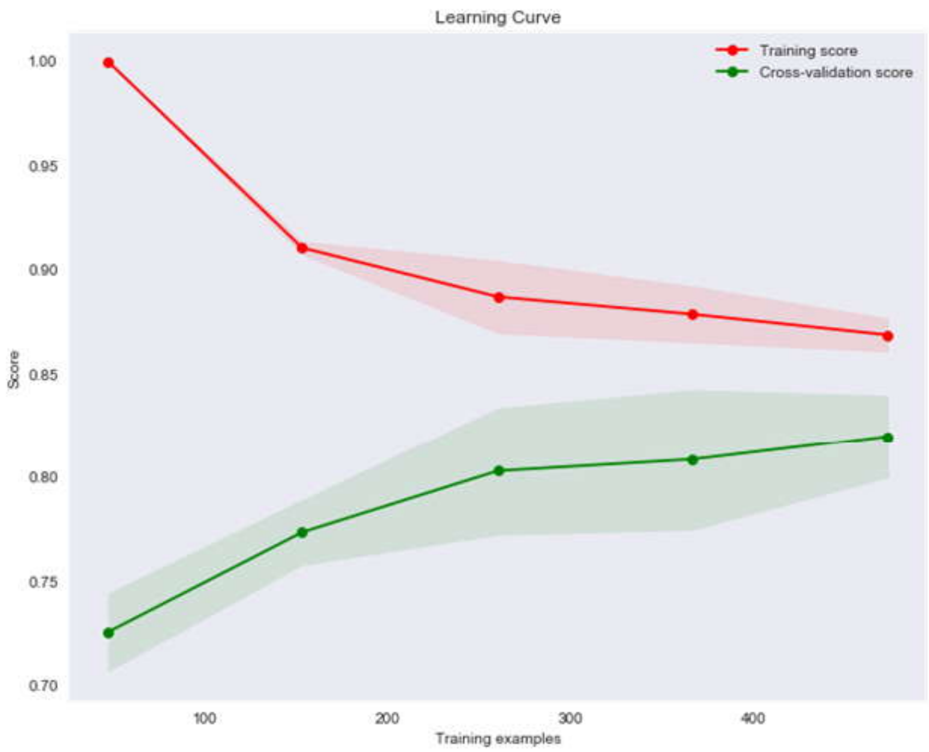
max_features = 3. Maximum 3 features can be considered at one time.

```
# We will build random forest on the same dataset.
random_forest_5 = RandomForestClassifier(max_depth=5,
n_estimators=100,max_features=3)
random_forest_5.fit(X_train,Y_train)

#Check on test data
Y_pred = random_forest_5.predict(X_test)
Y_pred_prob = random_forest_5.predict_proba(X_test)
print(accuracy_score(Y_test,Y_pred))
```

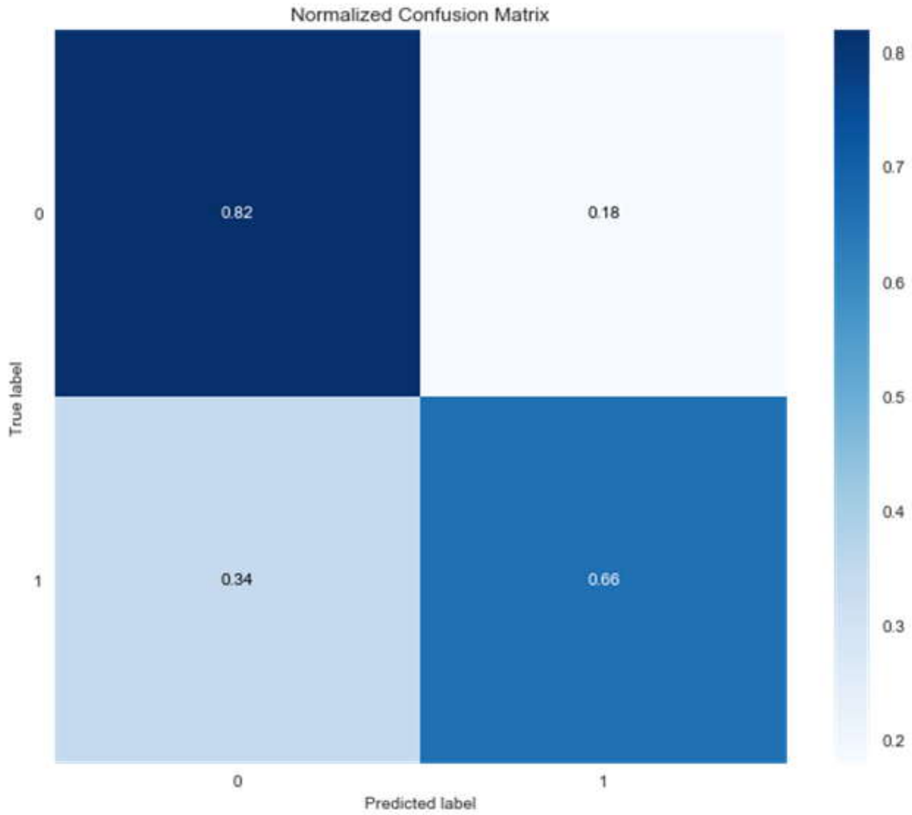
Learning Curve:

```
scikitplot.estimators.plot_learning_curve(random_forest_5,
X_train,Y_train)
```



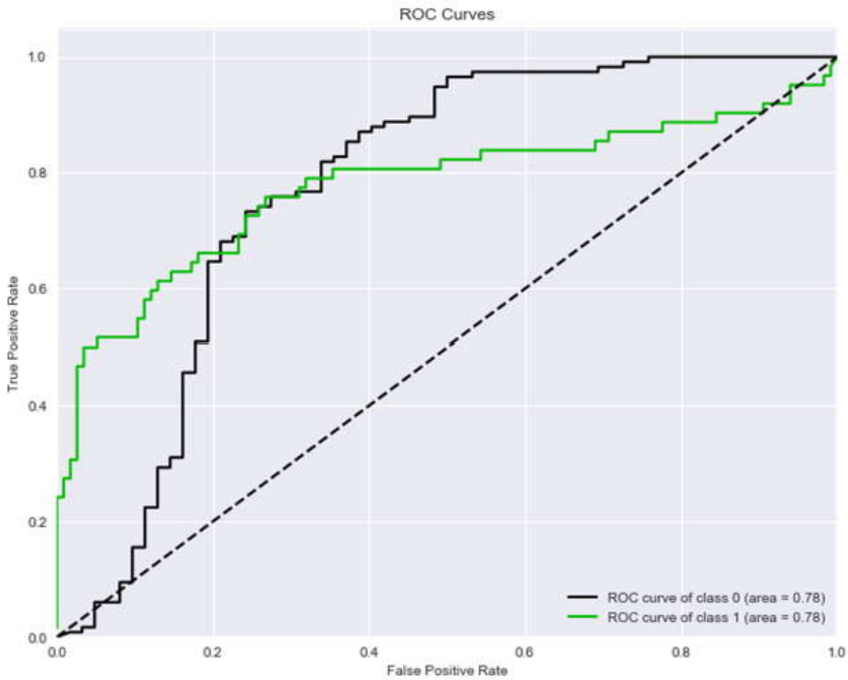
Confusion matrix:

```
scikitplot.metrics.plot_confusion_matrix(Y_test, Y_pred,
normalize=True)
```



ROC Curve:

```
Y_pred_prob = random_forest_5.predict_proba(X_test)
class_1_prob = list()
for i in Y_pred_prob:
    class_1_prob.append(i[1])
print(roc_auc_score(Y_test,class_1_prob))
model_result['Random Forest(Max Depth = 5)'] = roc_auc_score
(Y_test,class_1_prob)
scikitplot.metrics.plot_roc_curve(Y_test, Y_pred_prob,curves
=['each_class'])
```



7.5 Naïve Bayes

Naïve Bayes algorithm is used for classification problems. It is highly used in text classification. In text classification tasks, data contains high dimension (as each word represent one feature in the data). It is used in spam filtering, sentiment detection, rating classification etc. The advantage of using naïve Bayes is its speed. It is fast and making prediction is easy with high dimension of data.

This model predicts the probability of an instance belongs to a class with a given set of feature value. It is a probabilistic classifier. It is called**naïve** because it assumes that one feature in the model is independent of existence of another feature. In other words, each feature contributes to the predictions with no relation between each other. In real world, this condition satisfies rarely. It uses Bayes theorem in the algorithm for training and prediction.

Bayes theorem is given as:

$$P\left(\frac{X}{Y}\right) = \frac{P\left(\frac{Y}{X}\right) P(X)}{P(Y)}$$

P(*X*|*Y*) is the probability of event *X* given that event *Y* is true. (**Posterior Probability**)

P(*Y*|*X*) is the probability of event *Y* given that event *X* is true. (**Likelihood**)

P(*X*) is the probability of event *X*. (**Prior Probability**)

P(*Y*) is the probability of event *Y*. (**Evidence**)

In words we can state that probability of *X* given *Y* is equals to Probability of *Y* given X multiplied by probability of *X* divided by probability of *Y*.

$$\text{Posterior probability} = \frac{\text{Prior Probability} \times \text{Likelihood}}{\text{Evidence}}$$

Let's understand this by using an example. Consider a deck of 52 cards. Our first task is to calculate the probability that the card contains a king.

Total number of cards = 52

Total number of King cards = 4.

Probability of a card is a king card = P(King) = 4/52 = 1/13

Now, our next task is to calculate the probability of a card is king given that card is a face card.

We can use bayes theorem in this case:

$$P\left(\frac{\text{King}}{\text{Face}}\right) = \frac{P\left(\frac{\text{Face}}{\text{King}}\right) P(\text{King})}{P(\text{Face})}$$

$P\left(\frac{\text{Face}}{\text{King}}\right) = 1.$ As we know if the card is king, it is a face card.

P(King) = 1/13 which we already calculated in the previous task.

P(Face) = 9/52 = 3/13.

Putting the values in the formula:

$$P\left(\frac{\text{King}}{\text{Face}}\right) = \frac{1 \times \frac{1}{13}}{\frac{3}{13}}$$

$$P\left(\frac{\text{King}}{\text{Face}}\right) = \frac{1}{3}$$

Now, consider the machine learning classification problem. We have k features with C classes. Our goal is to predict the probability of an instance belong to a class Ci. Naïve Bayes algorithm takes the feature vector and calculates the conditional probability of class Ci with these set of features. We do it for all classes and the class having maximum probability is reported as predicted class.

Advantage of Naïve Bayes algorithm is easy and fast to implement. It performs well in the presence of categorical features. For numerical features data is assumed to come from normal distributions.

If the categorical feature contains some new category in test data, this model assign that as zero probability. We apply Laplace transformation to handle this problem. Also, assumption of independent prediction is present rarely in real world.

Let's build the model using naïve bayes algorithm:

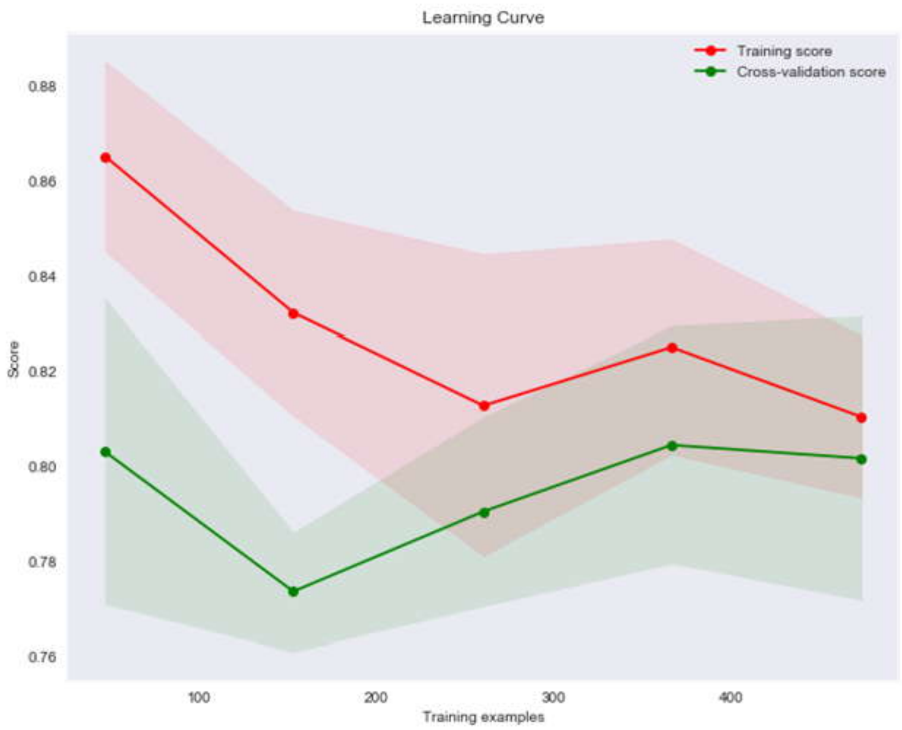
```
naive_bayes = GaussianNB()
naive_bayes.fit(X_train,Y_train)
```

Check on test data:

```
Y_pred = naive_bayes.predict(X_test)
Y_pred_prob = naive_bayes.predict_proba(X_test)
print(accuracy_score(Y_test,Y_pred))
```

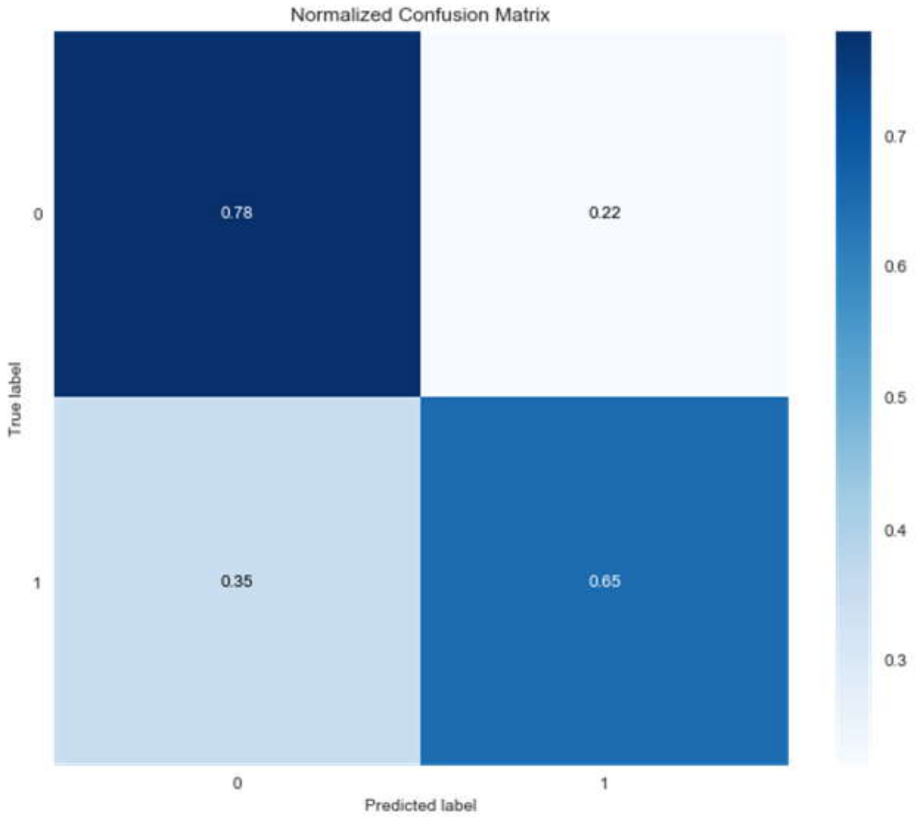
Learning Curve:

```
scikitplot.estimators.plot_learning_curve(naive_bayes,
X_train,Y_train)
```



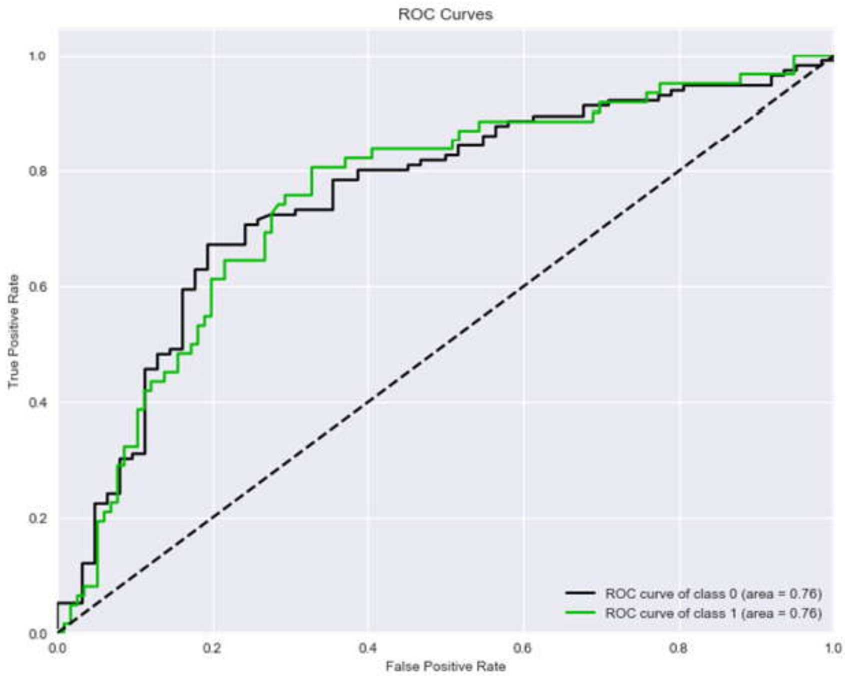
Confusion matrix:

```
scikitplot.metrics.plot_confusion_matrix(Y_test, Y_pred,
normalize=True)
```



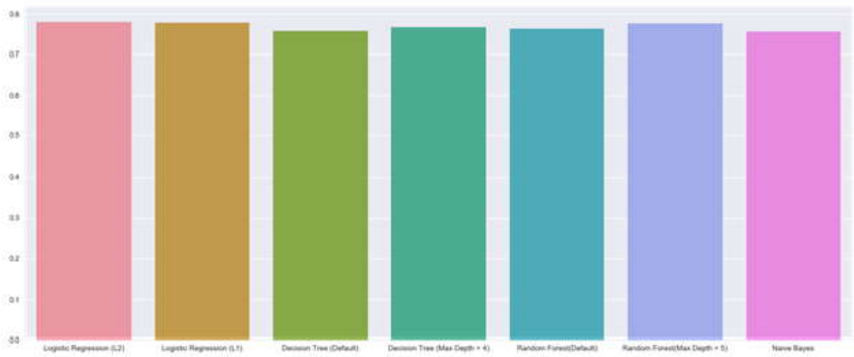
ROC Curve:

```
Y_pred_prob = naive_bayes.predict_proba(X_test)
class_1_prob = list()
for i in Y_pred_prob:
    class_1_prob.append(i[1])
print(roc_auc_score(Y_test,class_1_prob))
model_result['Naive  Bayes']  =  roc_auc_score(Y_test,
class_1_prob)
scikitplot.metrics.plot_roc_curve(Y_test,  Y_pred_prob,
curves=['each_class'])
```



Seaborn plot:

```
rcParams['figure.figsize'] = 20, 8
seaborn.barplot(x=list(model_result.keys()),y=list
(model_result.values()))
```



From above barplot we can see decision perform less as compare to logistic regression. But when we change the initial parameter by setting od decision tree, results are improved. Results are further improved by random forest. Naive bayes does not perform better in this case because the features are not completely independent with each other.

Chapter 8

Un Supervised Learning

In previous chapters we worked on supervised learning problems. We were given a set of features or attributes $x_1,x_2,x_3 \ldots x_n$ and a dependent variable y. Our task was to create a model to efficiently predict y. y can be categorical (in case of classification) or numerical (in case of regression). We also defined loss function for our model assessment.

In this chapter, we will focus on another branch of machine learning which is called un supervised learning. In un supervised learning problem outcome / dependent variable y is missing from data. We have to find properties from feature set x. As in supervised learning, we have y, model assessment is done by loss function. But in unsupervised learning as y is not available, we have to build algorithm that finds suitable properties based on the feature set x. These are inferences gathered from data without the response variable. So, there is no labeled data available in unsupervised learning, hence it is difficult to commit on performance of our model.

Unsupervised learning has a lot of use cases in today's world:

1. Finding the optimal location for a telephone tower. Different type of distribution exists in different part of the world. Clustering algorithm can group people and determine center to place the network tower.
2. Finding genetic sequence and patterns.
3. Create group of customers to give different offers and deals (Marketing segmentation).
4. Give summary of news and articles.
5. Finding patterns, interdependency of variables in the dataset.

8.1 Clustering

Clustering is the way to group the objects in a way that similar object formed/lies in single group. Number of groups may vary depending on the requirements.

Clustering is done to reveal interesting groups in the data. To understand the requirement of clustering, consider a set of customers. A credit card company ran a campaign and collected data from new customer including their age, gender, address, salary, designation, family, contact number, email etc. These details are gathered by their marketing executives by meeting persons at different part of the city. Now after collecting the data, the biggest task is to convert this contacts into their customer. One simple approach is their sales person start contacting each

person, tell benefits about their credit cards and tries to sell them. Customer listens for a few mins and started getting confused about the types of credit cards offers and hangs up the phone showing no interest in proceeding further.

Now, machine learning helps in these kinds of problem to convert this contact to customer. First it takes all the data (profile of customer, credit card types, their properties and past customers) and group the customers according to their profile. So, the algorithm gives output like this:

{Gender:Male, Age:20-30, Location: New Delhi} => Gold Signature Credit Card + 10% discount on movie tickets

{Gender:Female, Age:30-40, Location: Pune} => Diamond Plus Credit Card + 15% discounts on 5-star hotels

Now, these types of cards are recommended for the customers directly based on their profiles and interests. So, instead of showing customers all types of cards and benefits, sales person can directly send email or do a phone call for correct type of card and sell it.

8.2 K Means Clustering

K means clustering, assigns data points to one of the K clusters depending on their distance from the center of the clusters. It starts by randomly assigning the clusters centroid in the space. Then each data point assign to one of the cluster based on its distance from centroid of the cluster. After assigning each point to one of the cluster, new cluster centroids are assigned. This process runs iteratively until it finds good cluster. In the analysis we assume that number of cluster is given in advanced and we have to put points in one of the group. Consider a scenario of a broadband company. Based on their market analysis and expert's knowledge, the feel that they had five types of customers and they can give deals to them according to their group. So, in this case K is equals to 5.

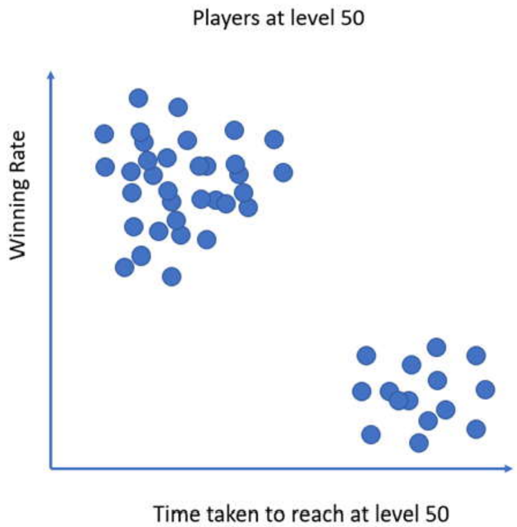
To understand K Means clustering take another example of online game playing. The game has a feature in which two players can compete and they can also play on predefined missions. Each time player completes a mission or wins in a battle he gets some points. After earning suitable points players are promoted to next level. Whenever player click on multiplayer game, systems tries to find player who is currently online and matches with the skill set of the player.

This problem can be solved easily by using the level of the player. If a player with level 5 request for a multiplayer game, system tries to assign him another online player with similar level. This approach seems great to start but it has some problem. Not all players are same. Some player understands the game easily and can perform better than others. For some it will take time to understand the features and use controls. In other words, some player can reach to level 50 at very fast speed, may be in few days. For others it may take months to get there.

Consider two players at level 50. Player A has very good understanding of game and handling of control and reached to 50 in 10 days. Another player B plays it for

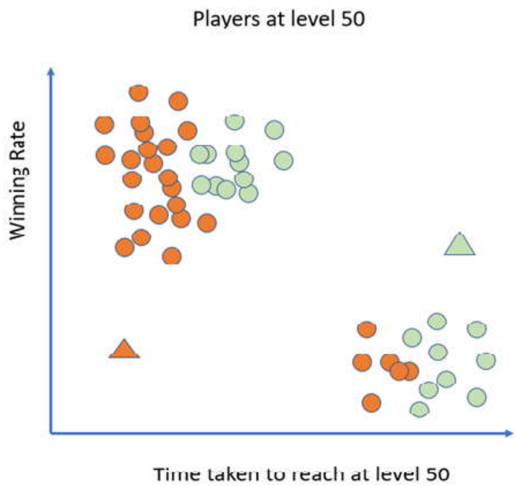
fun, don't think about moves or special powers much and reaches to level 50 in 4 months. If both player requested for multiplayer game, it is very high chances that player A defeats player B early. Player A will not like the game because it is very easy for him. Player B also don't like it because he is not able to win in multiplayer battles most of the time. If system finds the player suitable as their game playing strategy, both player can enjoy the game. Here comes the requirement of clustering.

If we divide the players in two groups based on their playing techniques and assign the player from their group, interest level will be maintained. Consider the scenario below:

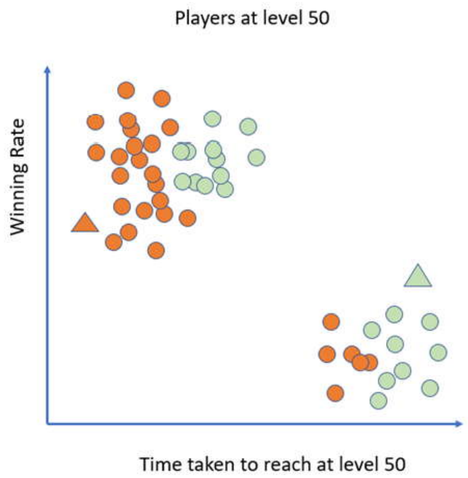


Some player take time to reach at level 50 and their winning rate is also low. Other player has high winning rate and they reach at level 50 very fast.

Now to perform K Means clustering, choose two points randomly and assign each player point to one of the two points based on distance.



Now, move cluster center to the center of clustered data.



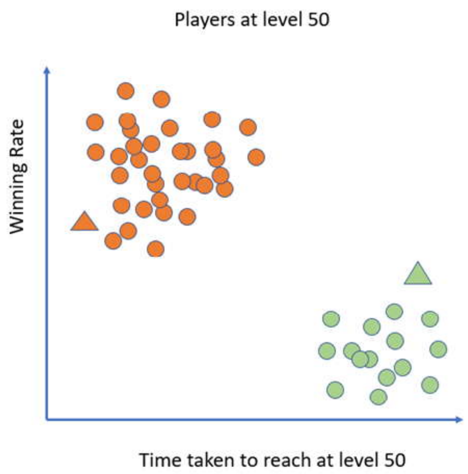
Now, again move each point to the nearest cluster center.

K Means clustering performs best data is well separated. When data points overlapped this clustering is not suitable.

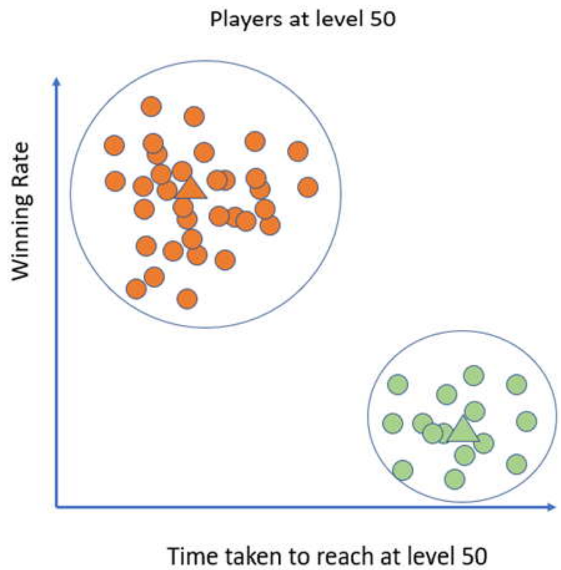
K Means is faster as compare to other clustering technique. It provides strong coupling between the data points.

K Means cluster do not provide clear information regarding the quality of clusters. Different initial assignment of cluster centroid may lead to different clusters. Also, K Means algorithm is sensitive to noise. It may have stuck in local minima.

K Means clustering is useful in many applications. One is placing mobile tower based on the usage of customer. Another is opening food store in different part of the city. Police can be assigned to different location based on crime rate and population. Hospitals can be opened in a way that it covers maximum part of the city.



Move cluster centroid to the center of the cluster data.



So, we can use training dataset to make the cluster. When we get instances from test data, we can assign them to one of the available cluster.

In some cases, K is not clearly defined, and we have to think about the optimal number of K. There are ways to do that. We will discuss that in upcoming sections.

When we talk about solving supervised learning problem, we think about cost function and tries to minimize cost of any assignment. In the case of clustering we also defined the cost function. Here, the cost function is not associated with any output y.

The cost of assignment of any point to any cluster is given as:

$$\sum_{k}^{K} \sum_{j \in Ck} \left\| Xj - Xk \right\|$$

Here, Xk is the centroid of the cluster. Xj is a single instance. We do it for all points in the cluster and for all the clusters. In K Means clustering our objective is to minimize the above value. Algorithm tries to shift the cluster centroids where the above value is lower with current assignment. After finding new cluster centroids again all the points are assigned to nearby cluster centroid and distance is calculated.

The algorithm can be understood as:

1. Start with K random centroids in the dataset
2. While there is change of cluster to any point
    a. For each point in the dataset
        (i) For each Cluster Centroid

1. Calculate distance between point and the respective centroid

(ii) Assign point to the cluster with closest centroid

b. For each cluster calculate mean

(i) Assign centroid to the mean of the cluster

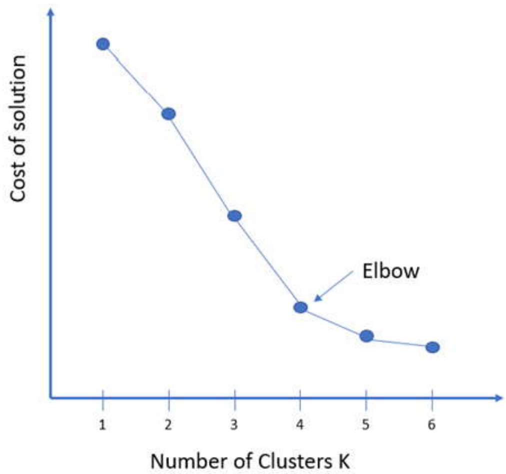
8.2.1 Problem with Random assignment of Cluster centroid

K Means clustering starts with assignment of cluster to random points. Now assume problem where points assigned to very near to each other. So, we start clustering algorithm and it ends with putting most of the points in one single cluster and approximately none in others.

To handle these kind of problem, we will start K Means clustering several times and choose the solution that gives minimum cost.

8.2.2 Finding value of K

Previously we built the algorithm for a fix value of K. In most cases value of K is unknown as there are more attributes and it is difficult to visualize number of cluster. Selecting the right value of K is very important. Wrong number of cluster do not help in solving the problem. In those situations, we use a technique called Elbow method to determine the optimal value of K. We use different value of K starting from 1 and keep increasing it. We plot a graph on value of K and cost of assignment. After some value of K, cost is not decreased significantly. On other words, plot shows an elbow as it is shown below. That value where elbow appears is taken as optimal value of K.



Let's work on KMeans clustering using python code example.

Import libraries

```
import pandas
import seaborn
import numpy
```

```
import scikitplot
from sklearn.cluster import KMeans,AgglomerativeClustering,
SpectralClustering

import matplotlib.pyplot as plt
%matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 10, 8
```

In this task we will use clustering_data_1.csv available in data folder

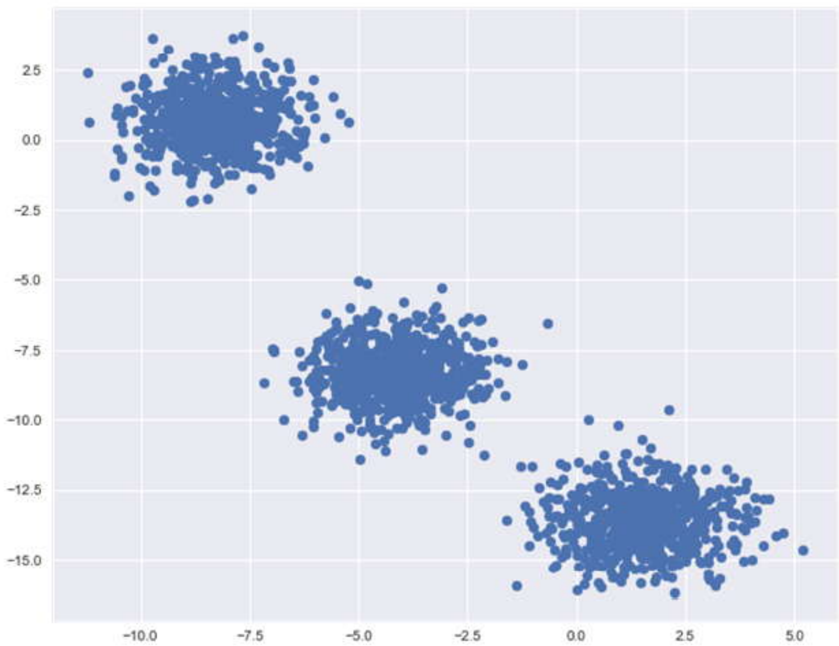
```
clustering_data_1 = pandas.read_csv('./data/
clustering_data_1.csv')
```

Check the shape of the data

```
clustering_data_1.shape
```

Data contains 2000 points with X and Y feature. We can plot this on 2D plane.

```
plt.scatter(clustering_data_1['X'],clustering_data_1['Y'])
```



From data we can see presence of tree clusters

```
#First create a KMeans clustering object and define number
of cluster = 2
kmeans = KMeans(n_clusters=2)
```

Fit KMeans clustering algorithm

```
kmeans.fit(clustering_data_1)
```

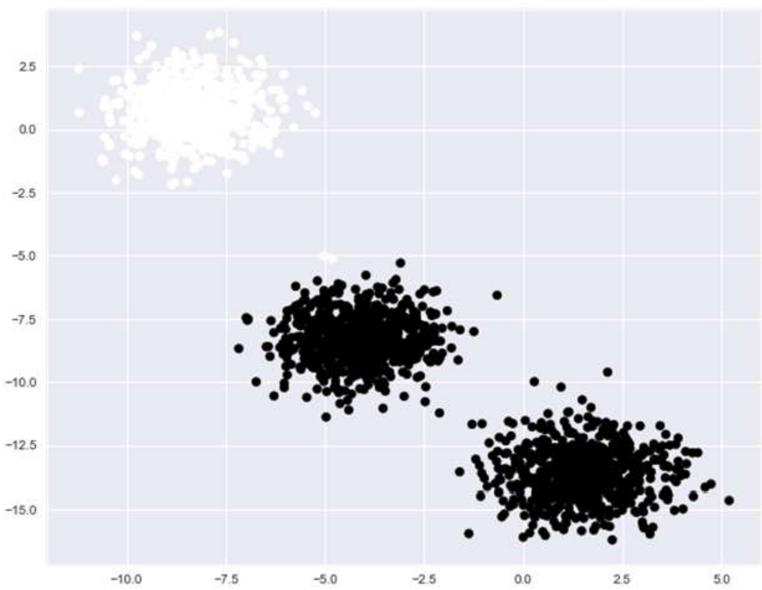
```
KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances=
    'auto',
    random_state=None, tol=0.0001, verbose=0)
                                                    In [8]:
```

Get cluster assignment

```
cluster_assignment = kmeans.predict(clustering_data_1)
```

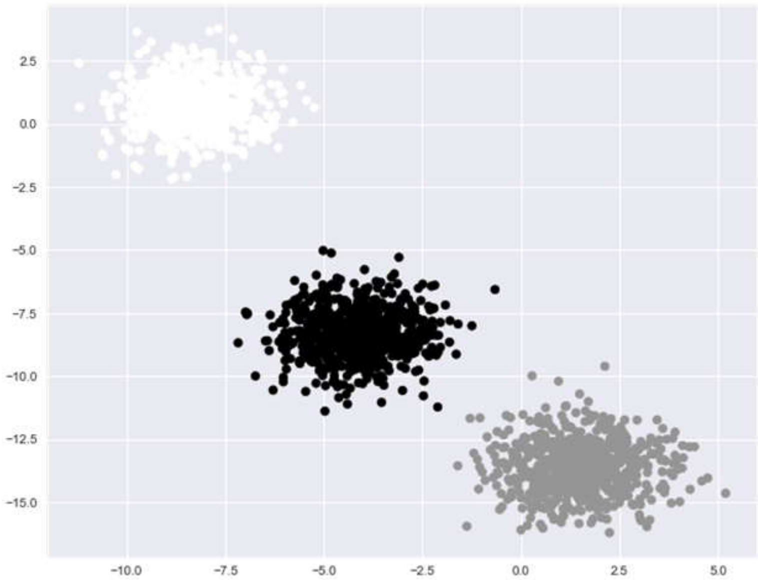
Plot cluster

```
plt.scatter(clustering_data_1['X'],clustering_data_1['Y'],
c=cluster_assignment)
```



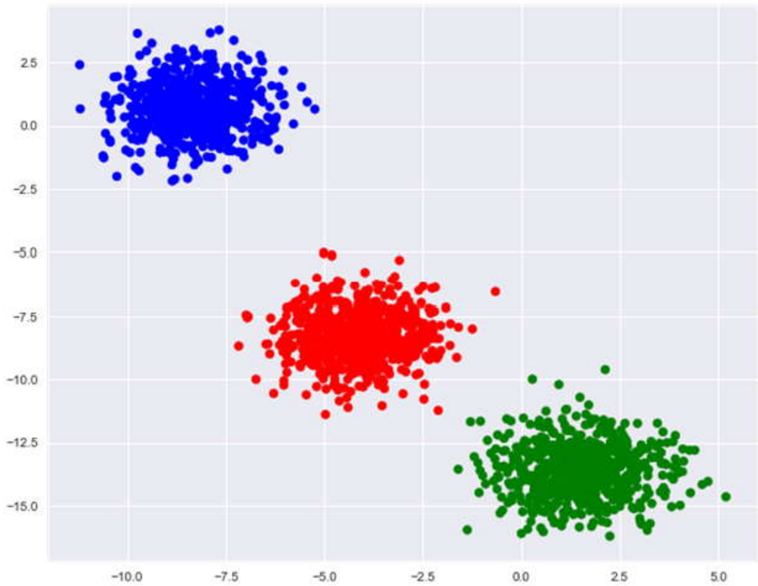
Above clusters are not looking appropriate. It is clearly visible that there are three clusters. Now let's try the same algorithm with K=3.

```
kmeans_3 = KMeans(n_clusters=3).fit(clustering_data_1)
#Get cluster assignment
cluster_assignment = kmeans_3.predict(clustering_data_1)
#Plot cluster
plt.scatter(clustering_data_1['X'],clustering_data_1['Y'],
c=cluster_assignment)
```



To make the assignment colorful, change the cluster assignment points.

```
cluster_assignment = cluster_assignment.astype(str)
cluster_assignment[cluster_assignment == '0' ] = 'b'
cluster_assignment[cluster_assignment == '1' ] = 'g'
cluster_assignment[cluster_assignment == '2' ] = 'r'
plt.scatter(clustering_data_1['X'],clustering_data_1['Y'],
c=cluster_assignment))
```



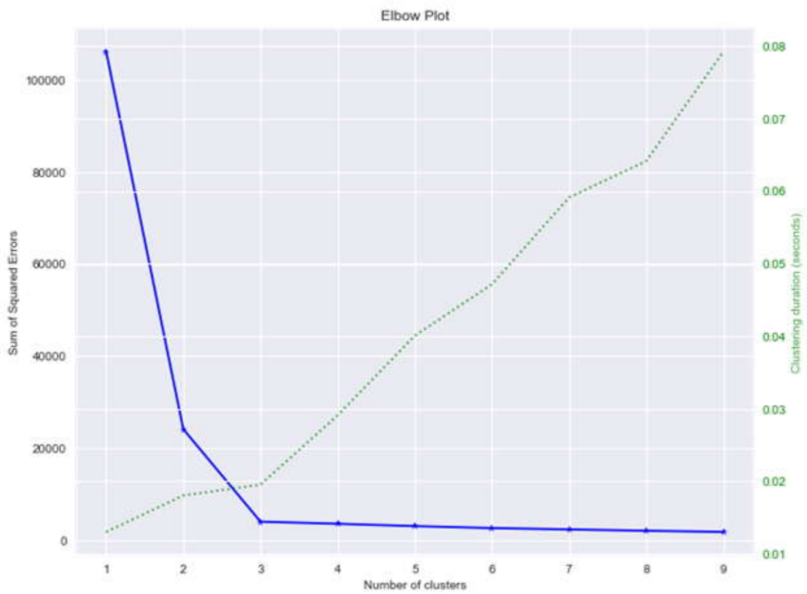
To make the assignment colorful, change the cluster assignment points.

```
cluster_assignment = cluster_assignment.astype(str)
cluster_assignment[cluster_assignment == '0' ] = 'b'
cluster_assignment[cluster_assignment == '1' ] = 'g'
cluster_assignment[cluster_assignment == '2' ] = 'r'
plt.scatter(clustering_data_1['X'],clustering_data_1['Y'],
c=cluster_assignment))
```

This is great. We can see three different clusters and our K Means algorithm found that correctly.

If we don't know what should be the correct value of K. scikitplot has a method to check the errors with number of cluster.

```
kmeans_elbow_check = KMeans()
scikitplot.cluster.plot_elbow_curve(kmeans_elbow_check,
X=clustering_data_1, cluster_ranges=range(1, 10))
```



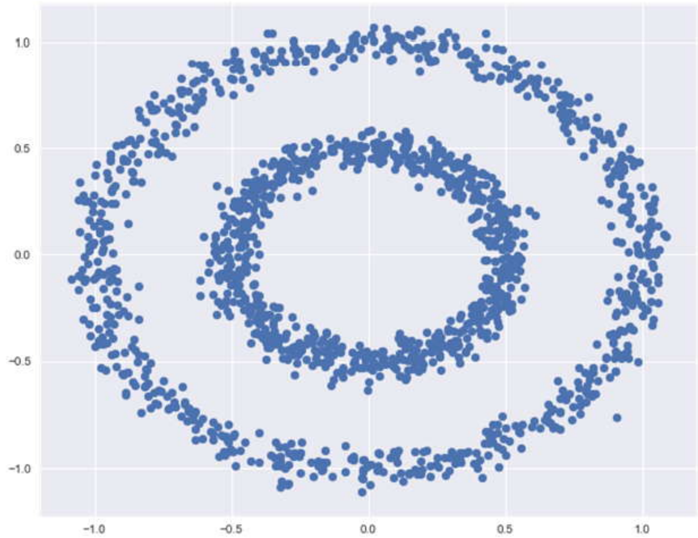
We can see that elbow is coming at 3. So, 3 is the correct choice for number of clusters.

Load another data

```
clustering_data_2 = pandas.read_csv('./data/
clustering_data_2.csv')
```

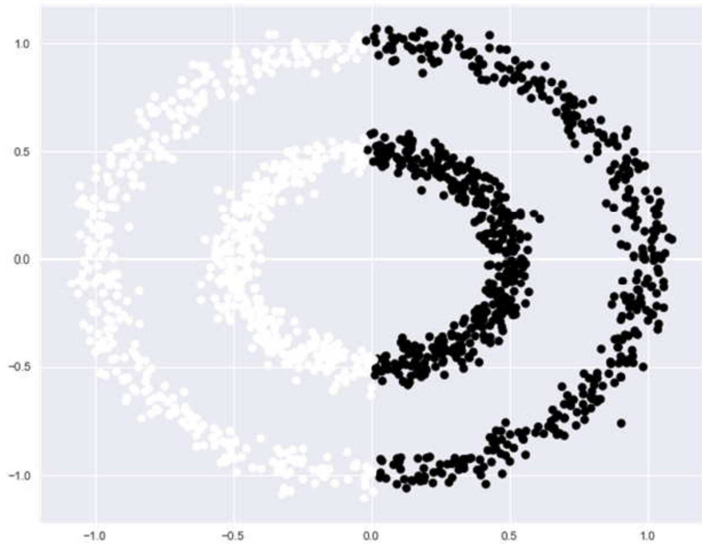
Plot the data

```
plt.scatter(clustering_data_2['X'],clustering_data_2['Y'])
```



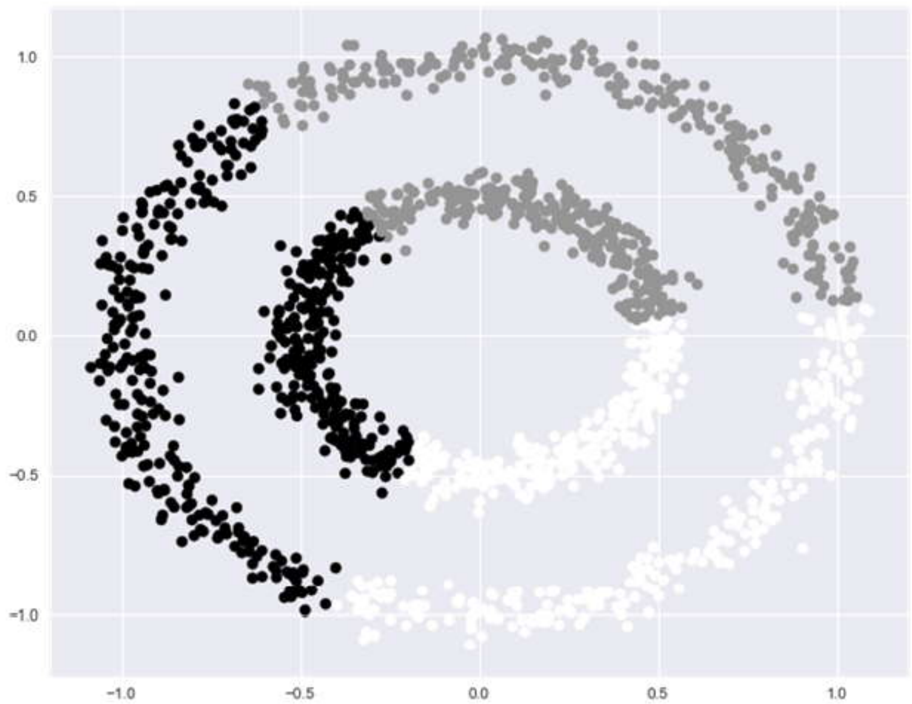
This is an interesting data. Let's apply KMeans clustering algorithm on it. We can see data is clustered in two group. One is inner circle and other is outer circle.

```
#First create a KMeans clustering object and define number
of cluster = 2
kmeans_2 = KMeans(n_clusters=2).fit(clustering_data_2)
#Get cluster assignment
cluster_assignment = kmeans_2.predict(clustering_data_2)
#Plot cluster
plt.scatter(clustering_data_2['X'],clustering_data_2['Y'],
c=cluster_assignment)
```



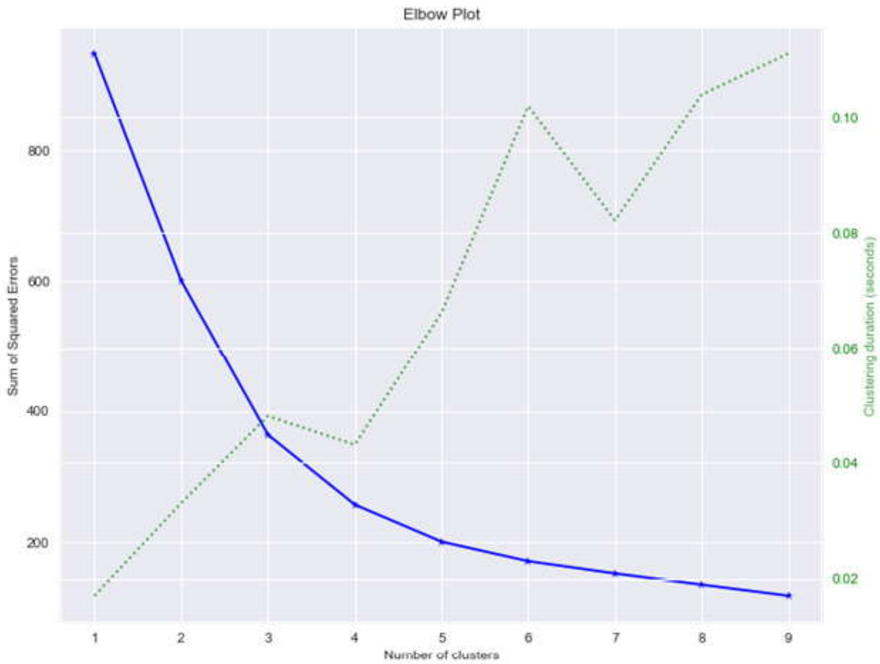
That is not something we want. Let's increase number of clusters.

```
kmeans_3 = KMeans(n_clusters=3).fit(clustering_data_2)
cluster_assignment = kmeans_3.predict(clustering_data_2)
plt.scatter(clustering_data_2['X'],clustering_data_2['Y'],
c=cluster_assignment)
```



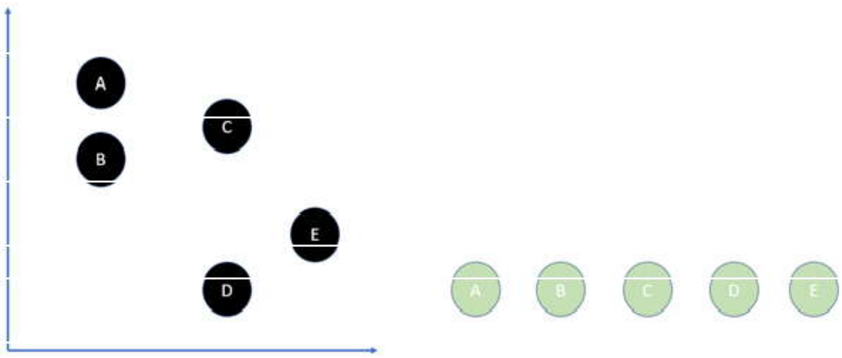
This is the limitation of this algorithm. It cannot find correct pattern in the above data.

```
#Try elbow method
kmeans_elbow_check = KMeans()
scikitplot.cluster.plot_elbow_curve(kmeans_elbow_check,
X=clustering_data_2, cluster_ranges=range(1, 10))
```

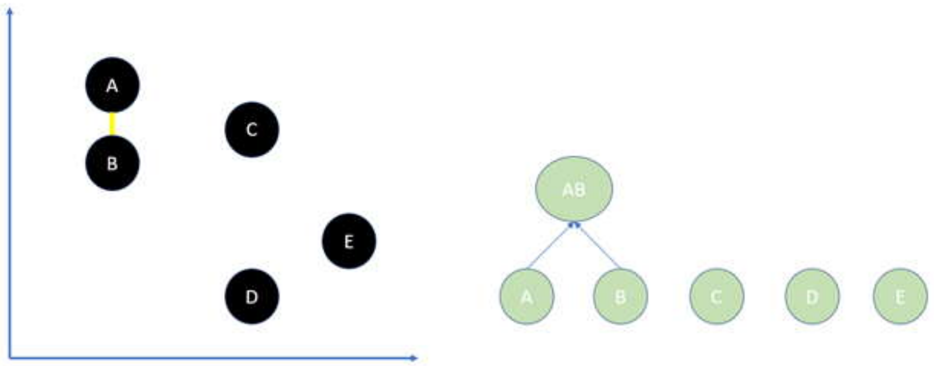


8.3 Hierarchical Clustering

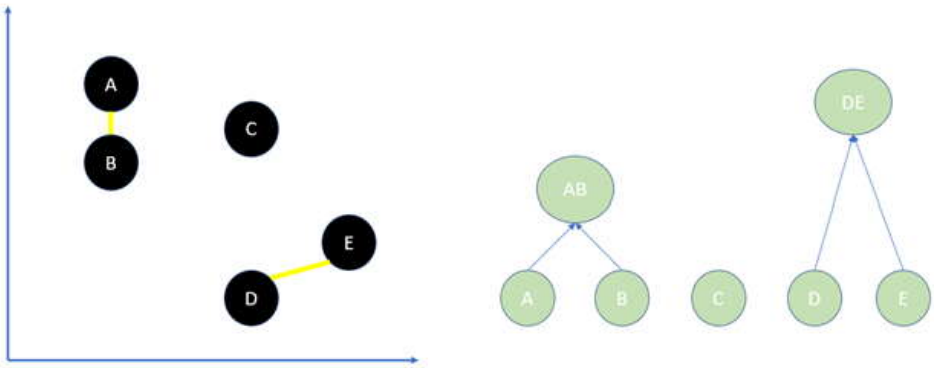
Hierarchical clustering is an algorithm that builds the hierarchy of clusters. In this clustering we make hierarchy of point by different strategy to achieve clustering. Consider the example of points in the space.



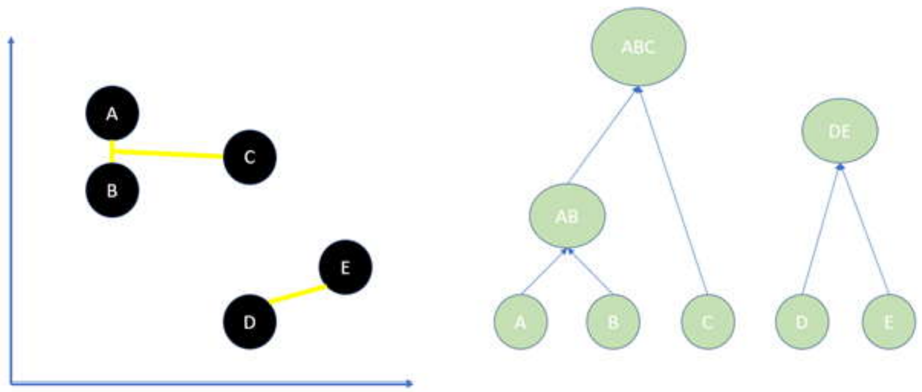
First consider each point as a cluster. So, at the beginning there are total 5 different clusters (A, B, C, D, E) are there. Now, merge the two clusters which are closet to each other.



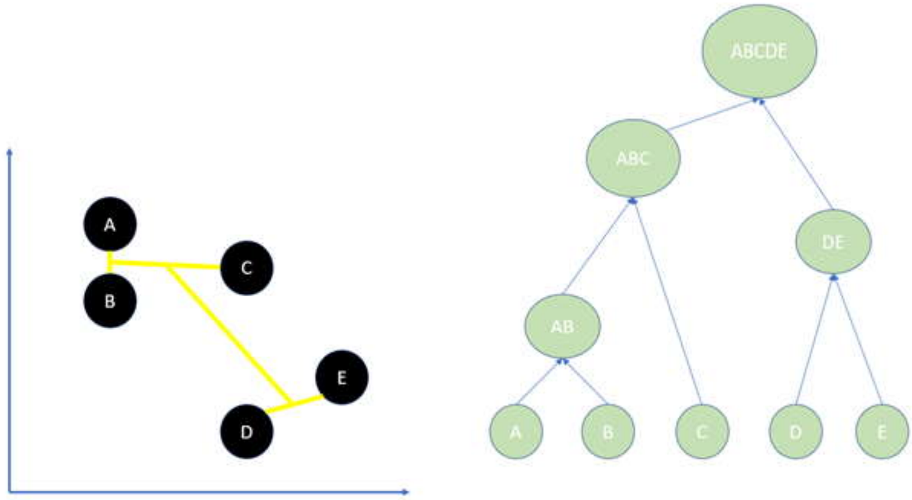
A and B are merged together. Now again find two nearest clusters among the four (AB, C, D, E) clusters. AB cluster center is considered at the middle of both the clusters.



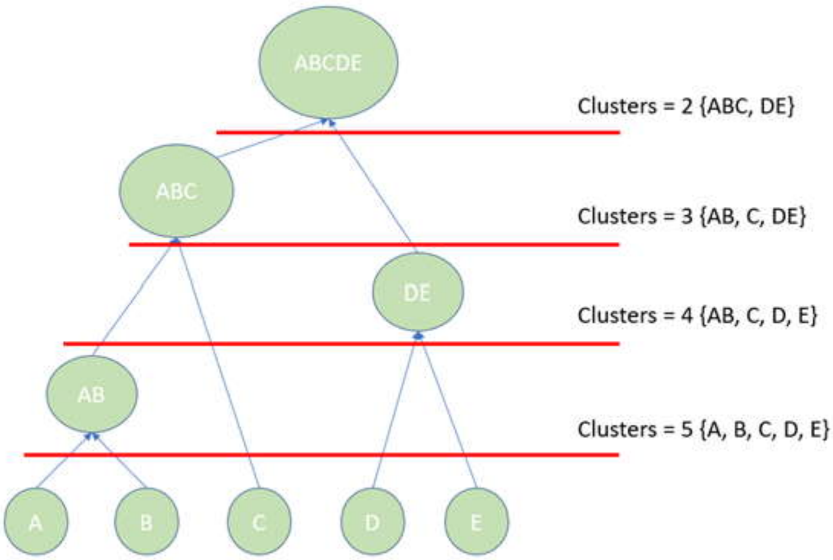
Again, do the same step.



Now make the last cluster,



After building the complete hierarchy we can cut the tree at different length and get the clusters.



We can decide the number of clusters by the tree. The general rule of thumb is to cut the tree at the middle of highest branch of the tree.

Hierarchical clustering also can't handle large amount of data. Clusters will be same on different run unlike K Means clustering which highly depends on initial assignment of cluster centroid. We need not to specify the number of cluster in advance. We can choose the assignment from tree.

Hierarchical clustering can be done by using two methods:

- **Divisive method (Top to Down approach):** In this technique, all the data is considered in one single cluster and then it is partitioned into two separate clusters. This step is performed repeatedly till we get the desired number of clusters or we create one cluster for each point.
- **Agglomerative method (Bottom to Up approach):** In this approach, we consider each point as a single cluster. In each step we select two closest point and merge them to make a single cluster. We perform this step repeatedly till only one cluster is left. This technique is described in the example above.

8.3.1 Distance Metrices

We can use different type of distance metrices to calculate the distance between points in hierarchical clustering.

$$\text{Euclidean Distance } \|x-y\|_2 = \sqrt{\sum_i (x_i - y_i)^2}$$

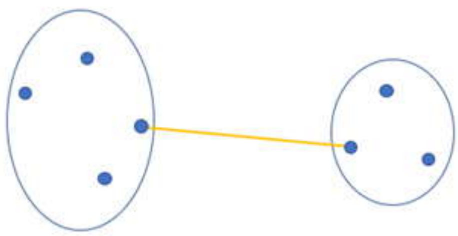
$$\text{Manhattan Distance } \|x-y\|_1 = \sum_i |x_i - y_i|$$

$$\text{Maximum Distance } \|x-y\|_\infty = \max_i |x_i - y_i|$$

8.3.2 Linkage

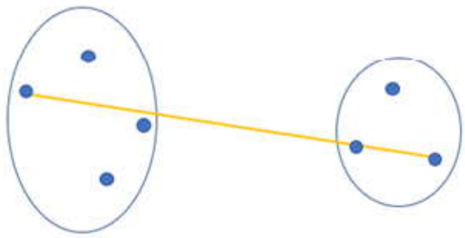
Different ways exist to define linkage between two clusters. Using different linkage techniques, we can get different cluster assignments.

- **Single Linkage:** Distance between two cluster is defined as the minimum distance between any points in both cluster. One point belongs to one cluster and second point belongs to another cluster.



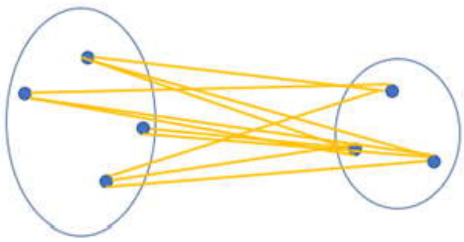
Single Linkage

- **Complete Linkage:** Distance between two cluster is defined as the maximum distance between any points in both cluster. One point belongs to one cluster and second point belongs to another cluster.



Complete Linkage

- **Average Linkage:** Distance between two cluster is defined as the mean distance between all points in both cluster. One point belongs to one cluster and second point belongs to another cluster.



Average Linkage

Application of hierarchical clustering are recommendation engine, social media analysis, market segmentation etc.

We used K Means clustering and it is not giving correct clusters. Let's try our next algorithm hierarchical clustering to get the clusters. We will use nearest linkage in this.

Create object of hierarchical clustering:

```
hierarchical_clustering = SpectralClustering(n_clusters=3,
affinity="nearest_neighbors")
```

Fit clustering model:

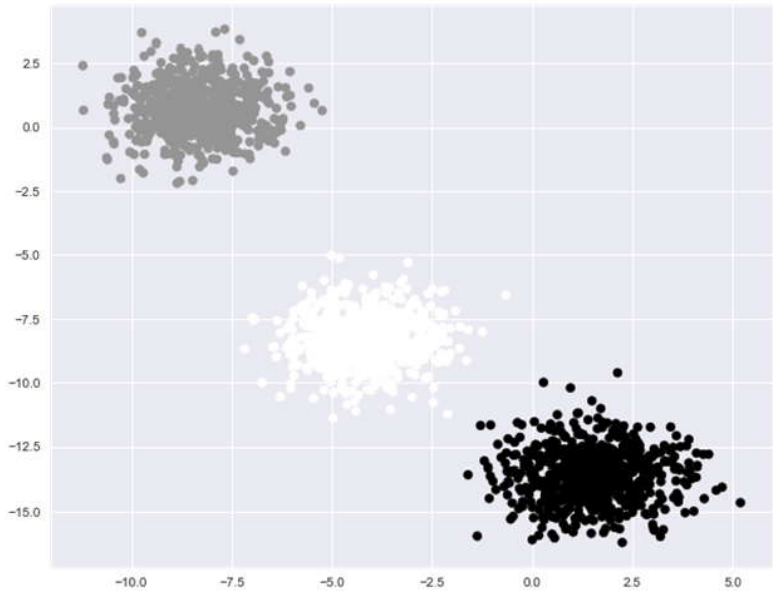
```
hierarchical_clustering.fit(clustering_data_1)
```

Assign cluster labels:

```
cluster_assignment = hierarchical_clustering.labels_
```

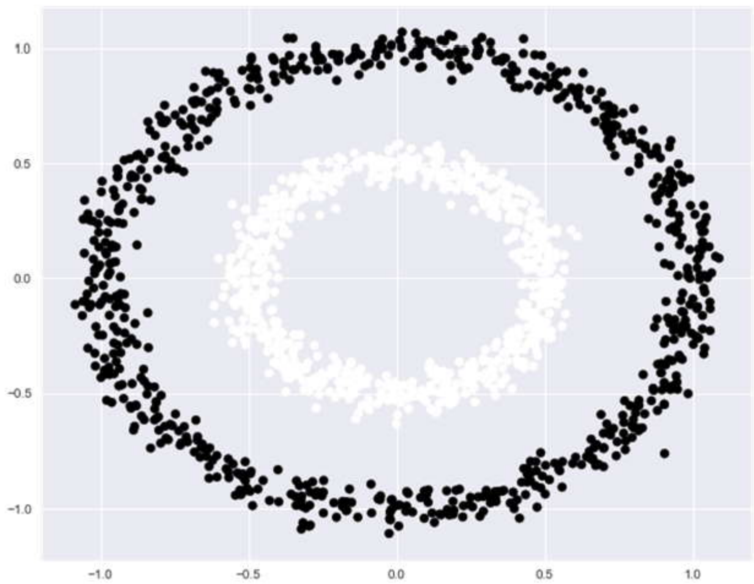
Plot clusters:

```
plt.scatter(clustering_data_1['X'],clustering_data_1['Y'],
c=cluster_assignment)
```



Hierarchichal clustering on clustering_data_2:

```
hierarchical_clustering = SpectralClustering(n_clusters=2,
affinity="nearest_neighbors")
hierarchical_clustering.fit(clustering_data_2)
cluster_assignment = hierarchical_clustering.labels_
plt.scatter(clustering_data_2['X'],clustering_data_2['Y'],
c=cluster_assignment)
```

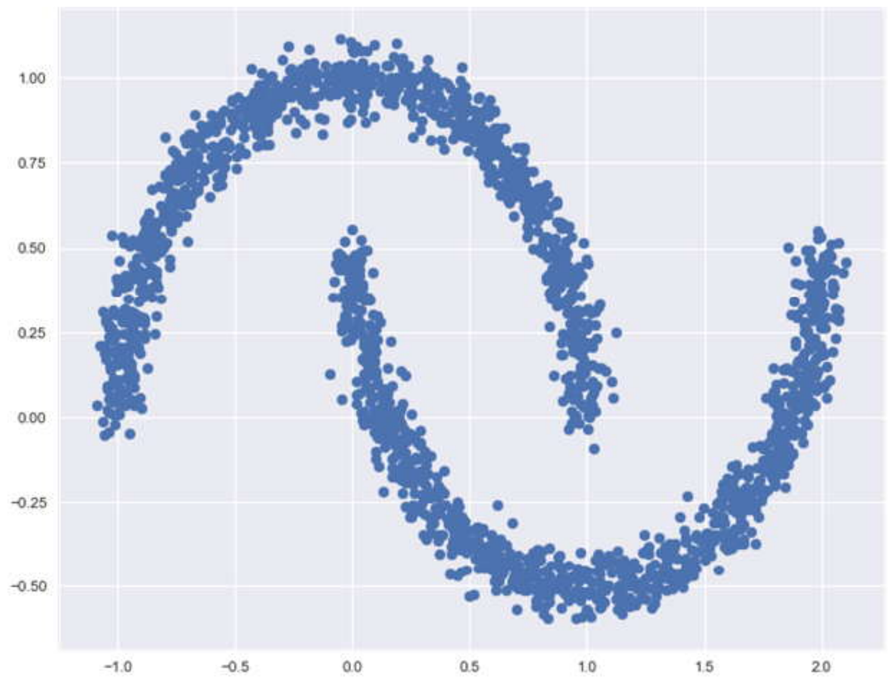


Hierarchical clustering performs better in these types of data as compare to K Means clustering. Let's do the experiment on another data.

```
#Load another data
clustering_data_3 = pandas.read_csv('./data/
clustering_data_3.csv')
```

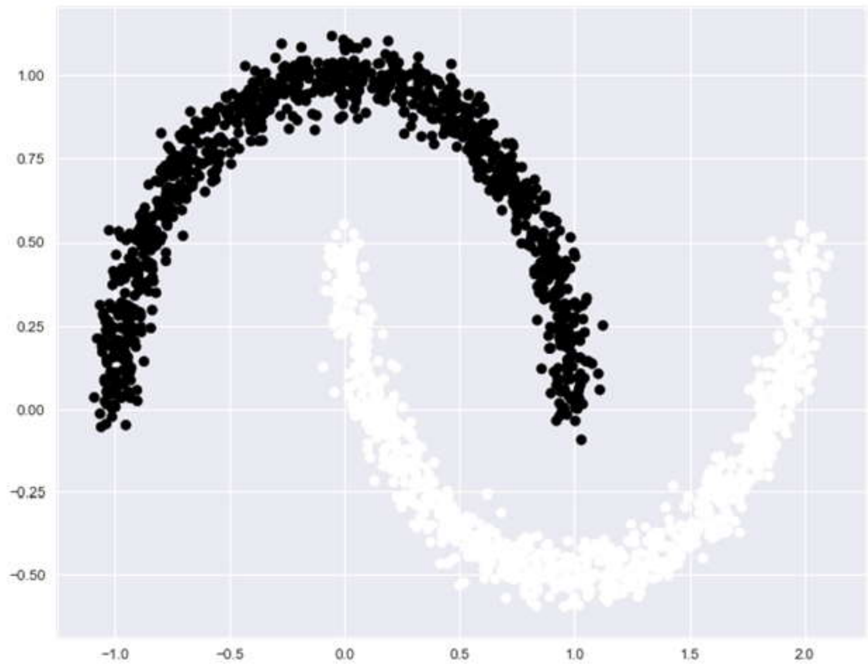
Visualize the data:

```
plt.scatter(clustering_data_3['X'],clustering_data_3['Y'])
```



Apply Hierarchical clustering:

```
hierarchical_clustering = SpectralClustering(n_clusters=2,
affinity="nearest_neighbors")
hierarchical_clustering.fit(clustering_data_3)
cluster_assignment = hierarchical_clustering.labels_
plt.scatter(clustering_data_3['X'],clustering_data_3['Y'],
c=cluster_assignment)
```



Hierarchical clustering performs better in these types of data as compare to K Means clustering.

Chapter 9

Text Analysis

We interact with text daily multiple times. First, we start by reading a newspaper in the morning. We start our Facebook or twitter account and start checking updates of our followings and friends. We send messages on WhatsApp or text message as SMS. In work we prepare task and send emails. We set reminders to complete day to day things. We read books and articles over the web. When we want to buy something, we visit the product selling page on Amazon or Flipkart, read about product specification and reviews before buying that product. We want to write some code. But facing some errors. We start searching the solution by typing our problem on search engine. Text is everywhere. Also, in the book which you are reading.

Text data grows exponentially. Today there is much more data as compare to five years back. Also, after five years, multiple times of data will be present over web as compare to current data.

With this huge amount of text, a lot of things can be done. We can understand the meaning of it. Identify and extract information from it. Search from document containing some information. Classify them into different groups. Identify topics hidden in the text and much more. In this chapter we will cover these interesting tasks with the text.

9.1 Basic Text Processing with Python

Text can be divided into different primitives.

- Document
- Sentences
- Words
- Characters

Document is a large collection of text. It contains Sentences. Each sentence is formed with a collection of words and each word form by characters. These are some general primitives of text which we will use in this chapter.

First, we will start with some basic text processing operations in python.

In python string can be declared by quotes or double quotes.

```
text1 = "The Vikram Sarabhai Space Centre is a space research
Centre of the ISRO, focused on rocket and space vehicles"
```

We can get length of strings by using len() function. It will return total number of characters in the string.

```
len(text1)
```

Output: 109

If we want to get number of words present in the string, we first need to split the string and count the words. This can be done by using split() function. Split function takes a separator by which it split the string in multiple words.

```
words1 = text1.split(" ")
len(words1)
```

Output: 19

We can also get the substring by providing indices with the string. Python uses indexing from 0. It means first character can be accessed by text1[0].

```
print(text1[4])
print(text1[4:10])
```

Output: 'V'

'Vikram'

String special operators: Different Operators can be used with strings to get different results.

Operator	Use
+	String Concatenation. Contact both left and right-side strings (Ex. A+B)
*	Repeats string by the number it multiplied (A*5)
in	Membership test. Returns true if character exists in the string. (Ex.- 'a' in B)
not in	Returns true if character doesn't exist in the string. (Ex. 'a' not in A)

9.1.1 String Comparisons

These functions return Boolean value (True/False) depending on the operation performed on string.

text.isalpha()	Returns true if string is not empty and contains only ASCII alphabets.
text.isalnum()	Returns true if string is not empty and contains only ASCII alphabets and numbers.

text.isdigit()	Returns true if string is not empty and contains only digits.
text.isdecimal()	Returns true if string is not empty and contains only decimal characters.
text.islower()	Returns true if string is not empty and contains only lower case characters.
text.isupper()	Returns true if string is not empty and contains only upper case characters.
text.isnumeric()	Returns true if string is not empty and contains only numeric characters.
text.startswith(start)	Returns true if string starts with the specified string
text.endswith(end)	Returns true if string ends with the specified string

```
#Get all words which has first character Capital
[wordCap for wordCap in words1 if wordCap.istitle()]
#Words with greater than length 5
[wordG5 for wordG5 in words1 if len(wordG5)>5]
```

9.1.2 String Conversions

These predefined set of functions are used for string conversions.

text.capitalize()	Returns string whose first character is capital and all other are small case.
text.title()	Returns string whose first character of each word is capital.
text.lower()	Returns string whose all characters are lower case.
text.upper()	Returns string whose all characters are upper case.
text.swapcase()	Returns string whose all lower-case characters are converted to upper case and all upper-case characters are converted to lower case.
text.casefold()	Returns casefold copy of current string.

9.1.3 String Manipulations

text.count(p)	Returns total number of occurrence of p in given string (text)
text.replace(old,new)	Replace all occurrence of old substring with new
text.find(p)	Return index of first occurrence of p in the given string.
text.rfind(p)	Return index of last occurrence of p in the given string.
seq.join(j)	Joins strings in the sequence with j
text.splitlines()	Returns strings which are separated by line separator in the original text.

text.lstrip([character])	Return string with character removed from its beginning if present. If no character specified, this method returns string with leading whitespaces removed.
Text.rstrip([character])	Return string with character removed from its end if present. If no character specified, this method returns string with trailing whitespaces removed.

Capitalize text

```
text1.capitalize()
```

Get in title form

```
text1.title()
```

Get in upper case

```
text1.upper()
```

Swapcase

```
text1.swapcase()
```

Casefold

```
text1.casefold()
```

Get index

```
text1.index('a')
```

Find from left

```
text1.find('a')
```

Find from right

```
text1.rfind('a')
```

Split lines

```
text1.splitlines()
```

lstrip

```
text1.lstrip("The")
```

9.2 Regular Expression

Regular expression is a powerful tool in any language to match the text patterns.

Python also supports regular expression. Python regular expression operation are supported by module re.

To use regular expression first we need to import 're' module.

```
import re
```

To perform regular expression search, we will follow this format

```
matchset = re.search(pattern,text)
```

Here, pattern refers to the rule we formed for matching and text contains string in which we want to perform the search. If search goes successful, match object is returned otherwise None.

We can consider this by following example:

```
text2 = "This news article is published on month:Jan"
matchResult = re.search(r'month:\w\w\w',text2)
if matchResult:
    print('Pattern exists ', matchResult.group())
else:
    print('Pattern not exists')
```

In above example we want to search for month followed by : and three characters. If it contains, result will be stored in matchResult object. We can print the result by using matchResult.group() method.

r which is used in the beginning of the pattern is to handle raw strings.

So, let's discuss some general rules used to make patterns:

1. Characters can be used as they appears. We can put them and they match directly (Ex. - a, B,4 etc.)
2. . (dot) is used to match with a single character.
3. \w is used to match any word character [a-zA-Z0-9_]. \W is used to match any non-word character.
4. \s is used to match single whitespace character (space, newline, tab). Similarly \S is used to match single non whitespace character.
5. \d is used to match single digit [0-9]
6. ^ is used to match start of the string.
7. $ is used to match end of the string.

Consider some example :

```
matchResult = re.search(r'de', 'abcdef') #Found, matchResult.
group() - de
matchResult  =  re.search(r'..cd',  'abcdef')  #Found,
matchResult.group() - abcd
matchResult = re.search(r'\w\w', '%%ab') #Found, matchResult.
```

```
group() - ab
matchResult = re.search(r'ef$','abcdef') #Found, matchResult.
group() - ef
matchResult = re.search(r'^ab','abcdef') #Found, matchResult.
group() - ab
```

We can also check for occurrence of this patterns.

+ is used to check one or more occurrence of the pattern.

* is used to check zero or more occurrence of the pattern.

? is used to check either zero or one occurrence of the pattern.

```
matchResult = re.search(r'de+f', 'abcdeef') #Found,
matchResult.group() - deef
matchResult = re.search(r'dk*e', 'abcdeef') #Found,
matchResult.group() - de
matchResult = re.search(r'e?f', 'abcdef') #Found, matchResult.
group() - ef
```

Square brackets can be used to match a set of words.

```
matchResult = re.search(r'&[ab]+', '&aef') #Found,
matchResult.group() - &ab
```

findall() - this function is used to find all occurrence of a specific pattern.

```
## Consider a tweet with hashtags
tweet = 'I am learning #datascience and it is awesome.
#python #machinelearning'

## Here re.findall() returns a list of all hashtags present
in the tweet.
hastags = re.findall(r'#\w+', tweet)
for tag in hastags:
   print tag
```

Output of above code is

```
#datascience
#python
#machinelearning
```

9.3 Natural Language Processing

Natural language is any language which is used for communication between humans. For example, Hindi, English, Spanish, German are languages in which human communicates. Language keep modifies itself. New word gets inserted. Old word start losing popularity in usage.

Different types of manipulation can be done on natural language text. Counting

words and their frequencies, part of speech tagging, parsing sentences, identifying entities, relationship between entities are different types of tasks covered in natural language processing.

To perform natural language processing with python we use nltk (Natural Language Toolkit). It is open source library written in python. It supports most of the natural language tasks.

We start by importing the library

```
import nltk
```

nltk has good number of text corpora. First, we need to download them to do some processing. This process will show a pop up on your screen and it will take time to download corpora from web. It is only a one-time task. From next time, we can skip downloading.

```
nltk.download()
```

Now to get a list of corpora, we can use following statement

```
from nltk.book import *
```

This statement shows list of corpora available. We can print any text.

```
text3
```

Output: <Text: The Book of Genesis>

We can check length of the text using len() statement.

```
print(len(text3))
```

To count unique words in the text

```
print(len(set(text3)))
```

We can count the frequency of words. In other way, given a word, how many times it appears on the given text. We can do it by using the following code.

```
freqCount = FreqDist(text3)
```

We can take the keys from freqCount and create a vocabulary.

```
vocabulary = freqCount.keys()
```

We can find out count of any word appears in the vocabulary

```
freqCount['The']
```

We can define some rule of frequency of the word. Suppose we want to take out words that has length 7 and occurs more than 50 times in the corpora

```
[word for word in vocabulary if len(word)>=7 and freqCount
(word)>=10]
```

9.3.1 Stemming

It is the process of finding the root word, hence reduce conflicts and convert words to their base words. For example, words like fishes, fishing, fisher all has the root word fish.

We can use porter stemmer available in nltk library.

```
fishwords = ['fish','Fishing','Fishes']
prt = nltk.PorterStemmer()
[prt.stem(ts) for ts in fishwords]
```

Output: [u'fish', u'fish', u'fish']

Stemming depends on the type of work. Sometimes it makes sense to do it, sometimes not. We have to take decision based on the problem statement.

9.3.2 Lemmatization

It is the process to convert words into their actual dictionary form. In nltk WordNetLemmatizer() is present to do this task.

We can understand the process by following code:

```
fishwords = ['fishes','Fishings','Fishes']
WNlemma = nltk.WordNetLemmatizer()
[WNlemma.lemmatize(ts) for ts in fishwords]
```

Output: [u'fish', 'Fishings', 'Fishes']

So, in the list are valid but fishes is lemmatized to fish.

9.3.3 Tokenization

This can be done by split() function available in python. But if we want to do it more clearly, we can use nltk tokenization.

```
text2 = "Why are you so intelligent?"
words = text2.split(' ')
print(words)
print(nltk.word_tokenize(text2))
```

Output:

['Why', 'are', 'you', 'so', 'intelligent?']

['Why', 'are', 'you', 'so', 'intelligent', '?']

Our first split function combines '?' with the previous word. But when we did it by using nltk, it create separate word.

Sometimes text is not well separated. We can use nltk sentence splitter to do the work. Consider the example below.

```
text3 = "His name John. He lives in U.K. with his wife Is he
the best? No he is not!"
sent1 = text3.split(".")
print(sent1)
sent2 = nltk.sent_tokenize(text3)
print(sent2)
```

Output:

['His name John', ' He lives in U', 'K', ' with his wife Is he the best? No he is not!']

['His name John.', 'He lives in U.K. with his wife Is he the best?', 'No he is not!']

First function just splits the text without considering their meaning. Second function is much better to extract meaningful sentences.

Another task that covered under Natural Language Processing is Part of Speech (POS) Tagging. This task basically tags the words in the sentence with noun, pronoun, verb, adjective etc.

First, we need to tokenize the sentence into words by using nltk tokenizer. Using nltk's pos_tag() we can do POS tagging.

```
tokenize_words = nltk.word_tokenize(text2)
nltk.pos_tag(tokenize_words)
```

Output:

[('Why', 'WRB'),
 ('are', 'VBP'),
 ('you', 'PRP'),
 ('so', 'IN'),
 ('intelligent', 'JJ'),
 ('?', '.')]

Part of speech tag is important to understand structures of sentences. We can group all the nouns together and form a custom dictionary.

Different notations are used for different part of speech.

Here is the list:

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
IN	Preposition or subordinating conjunction

JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
POS	Possessive ending
PRP	Personal pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WRB	Wh-adverb

We can write our own grammar rules and apply them using nltk.

9.4 Text Classification

Classification refers to classify data in some pre-defined categories. In earlier chapter, we worked on different classification algorithms and classify the data by them. In text classification domain, text is the data. Classify text in one of the given category is referred as text classification. Given a text we can classify it in one of the category like sports, movie review, business news etc. We can find about sentiment of comments to understand the product review. We can also correct spellings based on context.

We understand from previous chapters that machine learning algorithms works best with great features. The better your feature, better is your model performance.

Extracting features from text is not straight forward. All the information needed to solve classification problem exists in text. We need to represent it in a meaningful way so that it can be used as good features for machine learning algorithms.

Text features can be created by using words present in the text. We can remove stop words (The,It) from the text which are less important and common for all the articles. We can format it (convert case) or perform stemming to convert them to their base form.

Also, we can consider properties of words to create features. For example, word is noun or verb, sentence is correct or not, structure of the sentence, sequence of words (double face) etc.

So, we can apply different classification algorithms. First, we start with Naïve Bayes algorithm and see how we can use this to get right answers.

Suppose, we have given some text and we want to classify it in one of the category - politics, sports and business.

Now we got the article with text - 'presidential election impact economy'. As we can guess the article related to politics. But it also contains some information that it can relate to Business category also. As for now we have to choose one category, we will go for the most likely - Politics.

So, to use the naïve Bayes classifier to understand and generate predictions we have to define its terms.

Prior probability refers to the percentage of each class in the training data. For example, in our training data Politics appears 50% of the time, we say it's prior probability is 0.50.

Posterior probability refers to probability of class 'Politics' if the word is 'Presidential'.

As we know for this from naïve Bayes:

$$\text{Posterior Probability} = \frac{(\text{Prior Probability} \times \text{Likelihood})}{\text{Evidence}}$$

We can use the same for our text data:

$$\text{Probability('Politics' | 'Presidential')} = \frac{\text{Probability ('Politics')} \times \text{Probability ('Presidential' |'Politics')}}{\text{Probability ('Presidential')}}$$

We can calculate the same for other categories also. Whichever value is highest, we can assign that article to that category.

We also have to consider that naïve Bayes algorithm consider features independence. So, we calculate probability of each class by

$$\text{Probability}(Y|X) = \text{Probability}(Y) \times \prod_{j=1}^{n} \text{Probability}(X_j|Y)$$

We calculate the class by calculating probability of each words given a class and multiply them together. Also, we multiply them with the Prior probability of class Y.

Prior probability and Likelihood can be calculating by iterating over the training data and count the occurrence of words.

Probability(Y) can be calculated as number of document which are classified as Y divided by total documents.

Probability(xj | Y) can be calculated as word xj appears in the document classified as Y divided by total number of Y documents.

For example

Total number of document = N

Total number of document having class politics = P

Total number of P documents having word 'Presidential' = X

Probability('Politics') = P/N

Probability('Presidential'| 'Politics') = X/P

In naïve Bayes we can use features in two ways:

1. **Binary feature:** whether word appears in a document or not (Boolean)
2. **Multinomial Features:** number of occurrence of a word.

So, depending on problem we choose different variants to follow.

Text classification can be done by using scikit-learn. nltk also has predefined method to perform the task.

To use scikit-learn Naïve Bayes classifier, first we need to import it.

```
from sklearn.naive_bayes import MultinomialNB
```

We will use the 20-newsgroup dataset available in scikit learn datasets. This contains 20,000 data records which is classified into 20 category each. We can import this dataset by the following code:

```
from sklearn.datasets import fetch_20newsgroups
```

This data is well separated in training and testing. We can use training data to train our model and test performance on test data

```
training_data = fetch_20newsgroups(subset='train', shuffle=
True)
```

We can check for labels in the training dataset

```
training_data.target_names
```

Output:

['talk.politics.guns',

'comp.sys.ibm.pc.hardware',
'sci.electronics',
'comp.windows.x',
'rec.sport.hockey',
'rec.motorcycles',
'sci.space',
'alt.atheism',
'comp.sys.mac.hardware',
'misc.forsale',
'rec.autos',
'comp.graphics',
'sci.med',
'sci.crypt',
'talk.politics.misc',
'comp.os.ms-windows.misc',
'talk.politics.mideast',
'talk.religion.misc',
'rec.sport.baseball',
'soc.religion.christian']

We can also check the training data

```
print(training_data.data[0])
```

We can convert these sentences available in training data to word vector. We will do this by CountVectorizer available in scikit-learn. First, we need to import it

```
from sklearn.feature_extraction.text import CountVectorizer
```

Now we need to create its object and fit the model.

```
count_vector = CountVectorizer()
count_vector.fit(training_data.data)
```

We need to transform the data by using fitted model above:

```
training_count = count_vector.transform(training_data.data)
print(training_count.shape)
```

We can check the shape of our matrix:

```
print(training_count.shape)
```

Output: (11314, 130107)

In above matrix we are storing only count of words. We can weight them based on importance. We can use TF-IDF to do that. We need to import TFIDF.

```
from sklearn.feature_extraction.text import TfidfTransformer
```

Create Object and fit

```
tfidf_model = TfidfTransformer()
tfidf_model.fit(training_count)
training_tfidf = tfidf_model.transform(training_count)
```

We need to create model object

```
NBModel = naive_bayes.MultinomialNB()
```

Fit model on training data

```
NBModel.fit(training_tfidf, training_data.target))
```

Now to check performance of our model we can use test data available in the same dataset.

```
testing_data = fetch_20newsgroups(subset='test', shuffle=True)
```

Convert Testing data in the correct format

```
testing_count = count_vector.transform(testing_data.data)
testing_tfidf = tfidf_model.transform(testing_count)
```

Now, we transform testing data in the correct form. We can predict the target using predict() function

```
predicted = NBModel.predict(testing_tfidf)
```

We can evaluate our model using accuracy score. First, we import it by using scikit learn.

```
from sklearn.metrics import accuracy_score
```

Now, use this function to get accuracy

```
print(accuracy_score(testing_data.target,predicted))
```

Output: 0.77389803505

So, our model achieved 77.38% accuracy on 20 newsgroup data with 20 classes. This is a pretty good number. Now, we will combine our knowledge of NLP with text classification and will try to predictive power of our simple Naïve Bayes model.

```
import nltk
```

We can create a set of stop words available in nltk.

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

Now, we can define a data_cleaner() function. We will use our nlp knowledge gained in previous section. First, we tokenize the sentence using nltk tokenizer. We will use port stemmer and transform each word to its base form. We will also remove stop words from document and consider words with only alpha numeric values (Remove all symbols $,# etc.).

```
def data_cleaner(raw_data):

    cleaned_data = list()
    for row in raw_data:
        stemmed_words = list()

        #Find meaningful tokesn
        tokens = nltk.word_tokenize(row)

        #Convert words using stemming
        prt = nltk.PorterStemmer()

        for token in tokens:
            if token not in stop_words:
                if(token.isalnum()==True):
                    stemmed_words.append(prt.stem(token) )

        sent = ' '.join(stemmed_words)
        cleaned_data.append(sent)
    return cleaned_data
```

Now, we will transform our training data using this function.

```
training_data_modified = data_cleaner(training_data.data)
```

We can check the output of training data

```
print(training_data_modified[0])
```

Output:

From lerxst thing subject what car organ univers maryland colleg park line 15 I wonder anyon could enlighten car I saw day It sport car look late earli 70 It call bricklin the door realli small In addit front bumper separ rest bodi thi I know If anyon tellm model name engin spec year product car made histori whatev info funki look car pleas thank IL brought neighborhood lerxst

Similarly, we can modify text data also

```
test_data_modified = data_cleaner(testing_data.data)
```

Now, we can transform the data and do training

```
count_vector.fit(training_data_modified)
training_count  =  count_vector.transform(training_data_
modified)
tfidf_model.fit(training_count)
training_tfidf = tfidf_model.transform(training_count)
NBModel = MultinomialNB().fit(training_tfidf, training_data.
target)
```

Similarly, convert test data and predict

```
testing_count = count_vector.transform(test_data_modified)
testing_tfidf = tfidf_model.transform(testing_count)
predicted = NBModel.predict(testing_tfidf)
```

Calculate accuracy score

```
print(accuracy_score(testing_data.target,predicted))
```

Output: 0.802841210834

This is great. By using NLP techniques, we increased accuracy of our classifier to **80.28%**. We can use other techniques and classification method to improve it further.

9.5 Topic Modeling

Semantic similarly is a procedure to group words which has similar meaning. We can use wordnet to get these grouping of words. Wordnet contains property of word which can be used for semantic similarity.

Wordnet arranges word in hierarchy. We can follow that and take words similar which are close in hierarchy. Different techniques are available to choose similar words. One technique is to choose whichever in top hierarchy. Another technique is to choose lowest common ancestor.

Wordnet are available in nltk which can be used to solve our purpose. First we need to import necessary package

```
from nltk.corpus import wordnet
```

Now we want to give the word with the sense. Here we take three words 'cat', 'lion' and 'dog' as noun and find their similarity.

```
cat = wordnet.synset(cat.n.01)
dog = wordnet.synset(dog.n.01)
lion = wordnet.synset(lion.n.01)
```

Calculate similarity:

```
cat.path_similarity(lion)
cat.path_similarity(dog)
```

We can see the similarity between cat-dog and cat-lion.

Topic modeling is the statistical technique of finding topics from the text. It is used to find hidden properties in the document. A document contains a lot of topic. For example, if we see words like correlation, mean, sum these belongs to Mathematics. So we can say in a document of a type (like mathematical, biological, physical) has some similar words, which can be used to classify any new document.

In a document multiple topic can be detected. For example, it uses 5 words related to biology and 15 words related to Statistics. It can belong to both. Maybe it's a study of biology with some statistical points in it.

Topic modeling is different from using regular expression or using rules to find topics. It uses clustering to group the word together and extract topics out of them.

Topic modeling has different use cases across industries. Blog website use topics to group similar blogs together and recommended by using new blogs. Research papers can be categorized into category and people can consider them directly when they are trying to research cross domain.

We can extract topics from text by using different techniques. One technique that we discussed previously TFIDF (Term Frequency Inverse Document Frequency) can be used to find strong topics from the word. It marks important words that acts as topics.

Another useful technique is Latent Dirichlet Allocation or LDA. LDA works backwards, it takes document and figure out topics which creates that document. It is a matrix factorization technique. First, we present all the information into document term matrix. Each row represents a document and column represents all the possible words. Each cell value represents count of that word in the document. We convert this matrix into two matrices - document - topic and topic - term. LDA iterates through each word in each document and tries to improve topic - term relation. After good number of iteration both matrix got information regarding distributions. This is the stopping or acceptance criteria for LDA.

There are number of parameters exists in LDA to tune and get better modeling. Number of topics needs to be extracted from corpus. Topic terms exists in each topic. Number of iteration or maximum passes needed for convergence.

Let's take an example

We will use sample of 1000 rows from 20 newsgroup data for this task

```
import random
from sklearn.datasets import fetch_20newsgroups
```

```
training_data = fetch_20newsgroups(subset='train', shuffle=
True)
data500 = random.sample(training_data.data, 500)
```

We will do cleaning same as we done before in text classification. We will use same cleaning function as previously. We will create list of words.

```
training_data_modified = data_cleaning(data500)
training_data_cleaned = [clean(doc).split() for doc in
training_data_modified]
```

We can use gensim library. Gensim is used to handle text data and convert corpus into document term matrix.

```
import gensim
from gensim import corpora
```

After it, we will create document term matrix

```
dictionary = corpora.Dictionary(training_data_cleaned)
doc_term_matrix = [dictionary.doc2bow(doc) for doc in
training_data_cleaned]
```

Create model using genism

```
Lda = gensim.models.ldamodel.LdaModel
```

Training LDA Model

```
ldamodel = Lda(doc_term_matrix, num_topics=3, id2word =
dictionary, passes=50)
```

Print each line which represent topic with word.

```
print(ldamodel.print_topics(num_topics=3, num_words=3))
```

Output:

[(0, u'0.010*"line" + 0.009*"subject" + 0.009*"organ"'), (1, u'0.006*"line" + 0.006*"would" + 0.006*"subject"'), (2, u'0.055*"max" + 0.025*"2" + 0.017*"q"')]

We can name each of the group based on our understanding.

Chapter 10

Neural Network and Deep Learning

Neural network and deep learning provides the best solution for the hottest problem in the industry in computer vision, speech recognition, natural language processing etc.

As in today's world data is increasing with a rapid speed. Generating insights from large amount of data is the power of deep learning. Deep learning is becoming popular recently with GPUs in the market, which can do computation much faster than traditional CPUs. Also, a lot of open source libraries (TensorFlow, Torch, Keras, Caffe etc.) are introduced which helps and get the best from deep learning models. In this chapter first, we will understand the basic computation by using vectorized programming. Then, we will understand the concepts behind traditional neural networks followed by deep learning models. We will use keras in python for building models.

Deep learning moved artificial intelligence to a different level. Deep learning based system (DeepMind AlphaGo) defeated Lee Sedol in the Go board game who was the ranked 1 player for this game around the world. Not only in the games, deep learning is beating humans and generating better predictions in a lot of industry. From advanced failure detection in manufacturing industry to diagnose and predict early stage cancer, deep learning is used everywhere.

These are some of the applications of deep learning:

- **Face Recognition:** Identify faces, sense emotions or feelings.
- **Speech recognition:** Identify and generate speech of a person.
- **Driverless cars:** Drive cars on the street with full safety.
- Read raw handwritten text, correct them grammatically and produce articles.
- Generate text or videos as per person's mood.
- Identify trending news and detect fake news, warnings in advance.
- Translate text from one language to another language by following all the language rules.
- Image generation and object detection
- Text summarization

Deep learning and machine learning are closely related to each other. The relationship can be understood by using following:



Deep learning refers to training a deep neural network. But first we focus on what is neural network.

10.1 Vectorization

In Deep learning, we get most advantage when the data is large. When we are doing computations and performing result, it is equally important that the accurate result comes with faster speed. Vectorization is a process that converts scaler program to linear program. Vectorized programs has an ability to perform multiple instruction in a single instruction.

In python, we use numpy library to do vectorization for array-based programming. It contains functions that perform operations by using vectorized programming.

Let's first understand this by using dot product. In mathematics dot product or inner product refers to the sum of product of corresponding entries in two sequences. For example, if two vectors are given like this:

A = [3, 5, 4]

B = [1, 2, 3]

Now dot product A.B is given by:

$A.B = 3 \times 1 + 5 \times 2 + 4 \times 3 = 3 + 10 + 12 = 25$

Dot product is very useful operation and used in a lot of mathematical calculations. Let's do this in python.

First import libraries

```
import numpy
import time
```

Now create two large arrays of random numbers

```
x = numpy.random.rand(100000000)
y = numpy.random.rand(100000000)
```

Now, first do the dot product by using for loops in python. We will calculate time by using different timestamps (Before and after the loop.) We will also print sum to verify the results.

```
time1 = time.time()
sum = 0
for i in range(len(y)):
    sum = sum+x[i]*y[i]
time2 = time.time()
print('Time taken by scaler program :',str(time2-time1))
print('Sum value :', str(sum))
```

Output:

('Time taken by scaler program :', '68.9219999313')

('Sum value :', '24999964.345')

Now, let's do the dot product by using numpy library dot function.

```
time1 = time.time()
sum = 0
sum = numpy.dot(x,y)
time2 = time.time()
print('Time taken by scaler program :',str(time2-time1))
print('Sum value :', str(sum))
```

Output:

('Time taken by scaler program :', '0.100000143051')

('Sum value :', '24999964.345')

We can see that both program calculates same value of dot product. But scaler program (using for loop), took ~69 sec. On the other hand, vectorized program completed in ~0.1 sec.

So, we understand how vectorization can help in reducing time in calculation. We will try to do as much as possible.

numpy provides a special array ndarray. It is a fast-multidimensional array which provides vectorized operation and broadcast capabilities.

Create numpy one dimensional array:

```
arrlist = [1,2,3,4]
arrNumpy = numpy.array(arrlist)
```

Multi dimension array:

```
arrlist2 = [[1,2,3,4],[5,6,7,8]]
arrNumpy2 = numpy.array(arrlist2)
```

Dimension and Shape of array:

```
#Print dimention of array
print(arrNumpy2.ndim)
```

```
#Print shape of array
print(arrNumpy2.shape)
```

Zero, ones and empty array:

```
#Create array of zeros
zero_array = numpy.zeros(10)
print('Zero Array ',zero_array)
#Create array of ones
ones_array = numpy.ones(10)
print('Ones Array', ones_array)
#Create empty array
empty_array = numpy.empty(10)
print('Empty Array', empty_array)
```

Generate numbers:

```
#Create array of numbers
#Generate numbers from 0 upto length 10
numpy.arange(10)
#Generate numbers between a range
numpy.arange(-5,14)
#Generate numbers with a fixed distance between them
numpy.arange(3,6,0.2)
```

Indexing and Slicing:

```
#Indexing and slicing on 1D array
array = numpy.arange(15)
print(array)
print(array[2])
print(array[2:5])
array[3:9] = 34
print(array)
```

Output:

[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]

2

[2 3 4]

[0 1 2 34 34 34 34 34 34 9 10 11 12 13 14]

```
array2 = numpy.array([[1,2,3],[7,8,9]])
print(array2)
print(array2[1])
print(array2[1][1])
print(array2[1,1])
```

Output:

[[1 2 3]

[7 8 9]]

[7 8 9]

8

8

More on array slicing:

```
array = numpy.arange(10)
array2 = array2 = numpy.array([[1,2,3],[7,8,9],[13,14,15]])
print(array[3:])
print(array2[1:])
print(array2[1:,1:])
```

Output:

[3 4 5 6 7 8 9]

[[7 8 9]

[13 14 15]]

[[8 9]

[14 15]]

Boolean Indexing:

```
names = ['Abhishek','Ankita','Vijay','Sonawa','Sadhna',
'Ankita']
names_array = numpy.array(names)

#Check where the condition is true
print(names_array=='Vijay')

#Include names which satisfy the condition
namelist = names_array[names_array=='Ankita']
print(namelist)

array = numpy.arange(15)
large10 = array[array>10]
print(large10)
```

Output:

[False False True False False False]

['Ankita' 'Ankita']

[11 12 13 14]

Fast Element wise functions on array:

```
array = numpy.arange(-5,5)

#Absolute values of element
numpy.abs(array)

#Square root of each element
numpy.sqrt(numpy.abs(array))

#Square
numpy.square(array)

#Exponent
numpy.exp(array)

#Logrithm
numpy.log(numpy.abs(array))
numpy.log10(numpy.abs(array))

#Sign
numpy.sign(array)

#Ceil and Floor
numpy.ceil(array)
numpy.floor(array)

#Checking for infinite
numpy.isinf(numpy.log(numpy.abs(array)))

arrayTrig = numpy.array([0,45,90,180,270,370])

#trigonometric functions
numpy.sin(arrayTrig)
numpy.cos(arrayTrig)
numpy.tan(arrayTrig)
numpy.sinh(arrayTrig)
numpy.cosh(arrayTrig)
numpy.tanh(arrayTrig)
```

Array operations:

```
array1 = numpy.arange(1,60,3)
array2 = numpy.arange(1,80,4)

#Add two array (element wise)
numpy.add(array1,array2)

#Substract two array
numpy.subtract(array2,array1)
```

```
#Multiply two array
numpy.multiply(array1,array2)

#Divide
numpy.divide(array2,array1)

#Power
numpy.power(array2,array1)

#return element wise maximum
numpy.maximum(array2,array1)

#return element wise minimum
numpy.minimum(array2,array1)

#Logical operations
numpy.logical_and(array1,array2)
numpy.logical_not(array1)
numpy.logical_or(array1,array2)
numpy.logical_xor(array1,array2)

#Sort array
numpy.sort(array1)
```

Statistical Functions:

```
#Sum of all elements
numpy.sum(array1)

#Mean
numpy.mean(array1)

#Standard Deviation
numpy.std(array1)

#Variance
numpy.var(array1)

#Maximum element
numpy.max(array1)

#Minimum element
numpy.min(array1)

#Index of max element
numpy.argmax(array1)

#Index of max element
numpy.argmin(array1)

#Cumulative product of elements
```

```
numpy.cumprod(array1)

#Cumulative sum of elements
numpy.cumsum(array1)
```

Set operations:

```
#Union of two array
numpy.union1d(array1,array2)

#Intersection of two array
numpy.intersect1d(array1,array2)

#Set different
numpy.setdiff1d(array1,array2)
```

Linear Algebra:

```
#Dot product sum
array1.dot(array2)

array2d1 = numpy.array([[1,2,3],[4,5,6],[7,8,9]])
array2d2 = numpy.array([[1,1,1],[2,2,2],[3,3,3]])

#Inverse of matrix
numpy.linalg.inv(array2d1)

#Transposne
array2d1.T

#Get diagonal of matrix
numpy.diag(array2d1)

#Compute determinant
numpy.linalg.det(array2d1)

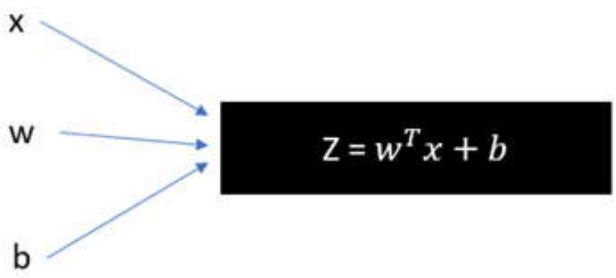
#QR Decomposition of matrix
q,r = numpy.linalg.qr(array2d1)

#Compute trace (sum of all diagonal element)
numpy.trace(array2d1)

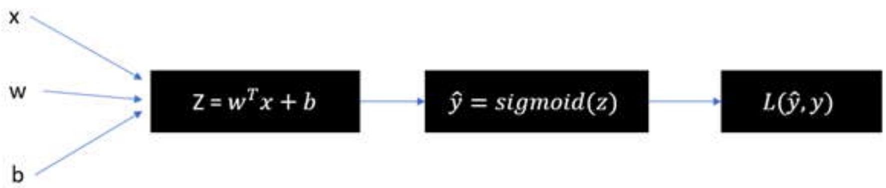
#SVD decomposition
a,b,c = numpy.linalg.svd(array2d1)
```

10.2 Neural Network

In earlier chapter, we discussed classification by using logistic regression. We give input x and computed parameters w and b. Here, is the structure of our logistic regression model.



We finally put this value in the sigmoid function and calculate our prediction. We keep track of loss for all the training example. We move the parameters in a way to reduce the loss. The complete structure of logistic regression is given as:



Now, let's move to neural networks. Neural networks are based on the theory of neurons.

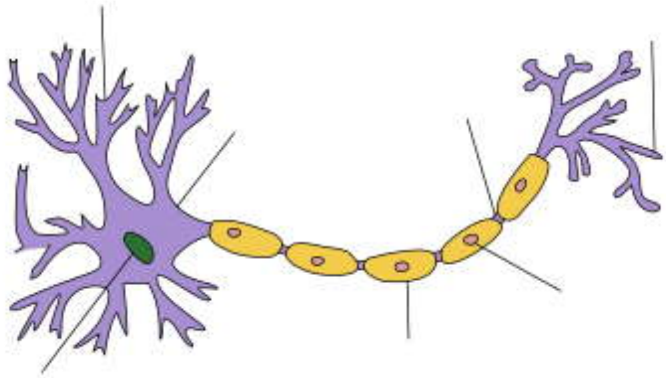
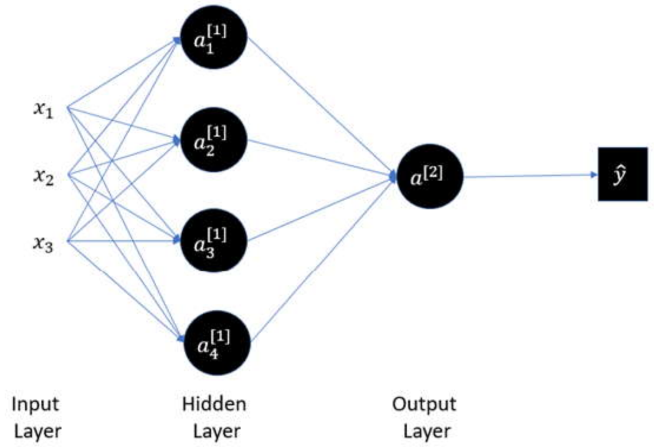


Image Source: wikipedia.com

A neuron is a cell which receive, process and transmit information (signal) to other cells. Neurons are connected with each other in a neural network. That is from where the term comes from. So, in our neural network model, nodes (similar to neurons) connects with each other and receive, process and pass the information.

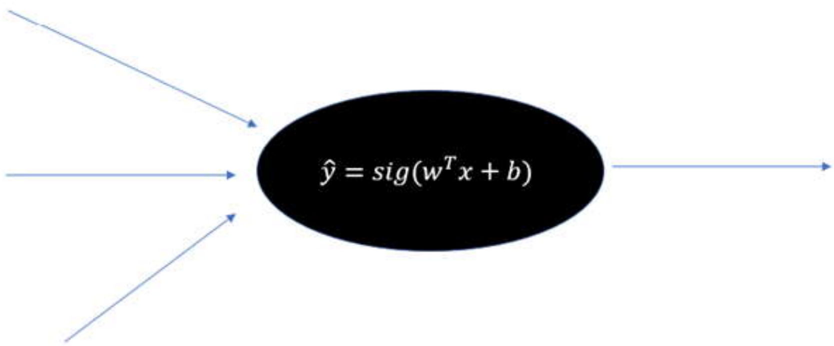
Neural networks can be described by using this:



Above example shows basic neural network with Input, Hidden and Output layer. Input layer contains input from training set. Hidden layer cannot be seen in training set. That's why we call it hidden layer. Output layer is associated with output.

Here $a^{[i]}$ represent *i* layer. Our first layer contains four nodes. It is connected with each input fields. Another layer is connected with layer 1. So, it is a 2-layer neural network.

Now, let's understand the processing in each node. In each of the node first parameters w and b is calculated, which calculates *z*. After it, sigmoid function is applied to get the output of the node $\hat{y}$.



In each node of the neural network (of all layers except input layer) same calculation is performed.

In the above example:

$$a_1^{[1]} = sig\left(w_1^{[1]T} x + b_1^{[1]}\right)$$

$$a_2^{[1]} = sig\left(w_2^{[1]T} x + b_2^{[1]}\right)$$

$$a_3^{[1]} = sig\left(w_3^{[1]T} x + b_3^{[1]}\right)$$

$$a_4^{[1]} = sig\left(w_4^{[1]T} x + b_4^{[1]}\right)$$

Also,

$$a^{[1]} = \left(a_1^{[1]} a_2^{[1]} a_3^{[1]} a_4^{[1]}\right)^T$$

We can also make the same equation for output layer. But Output layer (2nd Layer) is connected with hidden layer and not with input layer.

$$a^{[2]} = sig\left(w^{[2]T} a^{[1]} + b^{[2]}\right)$$

The complete process on a single node works like this:

- Input is given to the neuron.
- Input is multiplied by the weights.
- Input with multiplied weights are summed and bias term is added.
- Activation function is applied.
- Output is generated.

We can understand this by using decision-making process. Suppose you want to spend some money to watch a musical concert. But we need to decide things on various factors which are:

1. Do you have enough money in your bank account?
2. Are your friends going?
3. Is it holiday?

Let's consider these three yes/no questions as input *x*1, *x*2 and *x*3. And weight of these questions as 5, 3, 4. If your score is greater than 6, you go, otherwise you move on. Let's assume your answers are yes, yes and no. Your overall score for this problem is:

Score: 5 × 1 + 3 × 1 + 4 × 0 = 8 (Greater than 6, Go don't miss it.)

Neural networks work in similar fashion. They have weights for different neurons as input. And neuron feeds information to another neuron.

10.2.1 Gradient Descent

Now the challenge is how to make neural network to learn from training data. Our goal is to create a model that generates output correctly for training data. In other words, our model should minimize the cost function.

$$C(w,b) = \frac{1}{2n} \left\| y - \hat{y} \right\|$$

Here, *w* is the set of parameters and *b* is bias. '*n*' denotes the number of training example. '*y*' represents actual output and $\hat{y}$ is the predicted output. '*C*' is a cost

function which is also known as Mean Squared Error (MSE). So, it is obvious that this function result cannot be negative. The best case is when $C(w) \simeq 0$. Our parameters and bias should be in a way that moves cost function towards zero.

We use gradient descent algorithm to do this work. We change the parameters in a way that reduce the cost and try to reach to global minima.

Now, it is important way by which we can improve our cost function. This is known as learning rate. It is the rate by which we modify our cost function in the direction to right results. Learning rate plays an important role. If it is lower, it takes time in convergence. If it is higher, we may miss the global minima. By this, we need to make choice based on the nature of the problem.

In Neural networks, we use backpropagation algorithm to compute gradient. In backpropagation we modify internal weights to generate desired output.

10.2.2 Activation function

Till now in our explanation of neural networks, we use sigmoid function as an activation function. The goal of activation function is to convert input signal to output signal.

$$a = \frac{1}{1 + e^{-z}}$$

This function generates value in the range of 0 to 1. We can use other functions also for activation.

First, start with very simple activation. Suppose we can start with the activation based on some threshold value. If the output of neuron is greater than some threshold we can consider it as 1 otherwise 0. That is good for binary classification. But this concept failed in multiclass classification where we have different classes and we want to take one from each neuron. Instead of giving Boolean answer about activation, it is good to say that we can choose the activation value and maximum value at next level.

Now consider another activation as:

ActLin = *Cx*

This activation function has linear relationship with input. So, activation is c times the input x. This helps the problem of multi class classification. Whenever there are more classes we can decide a range for activation and choose the class accordingly.

Everything seems fine, but there is a problem. The gradient of this function is c. In other words, gradient is constant. It means in another layer, we can completely represent it as function of input and subsequent in all layers. So, our output is just a function of input. This means the power of using multiple layers is out.

The intuition behind neural network is to represent any function. Using nonlinear functions, we can learn complex and nonlinear relationship between input and output.

Now let's discuss about some popular activation functions

10.2.2.1 Sigmoid Function

Sigmoid function is also known as logistic function. For x, value of function is 1/(1+exp(-x)). This function is smooth and non-linear in nature. Combination of these function is also non -linear which is good for our neural network model. It bounds the output in [0,1] for any input (-inf,inf). Let's see the example of graph in python for sigmoid function.

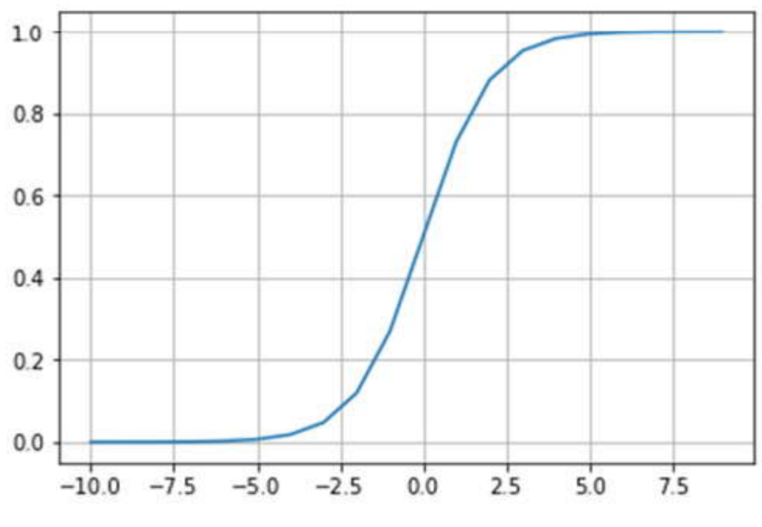
```
#Declare sigmoid function
def sigmoid_array(x):
    return 1 / (1 + numpy.exp(-x))
```

Generate numbers from -10, 10 with step 1.

```
numbers = numpy.arange(-10,10)
sigmoids = sigmoid_array(numbers)
```

Generate graph

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(numbers, sigmoids)
ax.grid(True)
plt.show()
```



As we can see from above figure, for higher values of input, output remains nearly constant (0 or 1). This function is mostly used in binary classification.

10.2.2.2 Tanh Function

Tanh function is represented by (exp(2x) – exp(-2x)) / (exp(2x) + exp(–2x)). It is also a nonlinear function, which generates output in the range (–1,1).

First define tanh function for numpy array

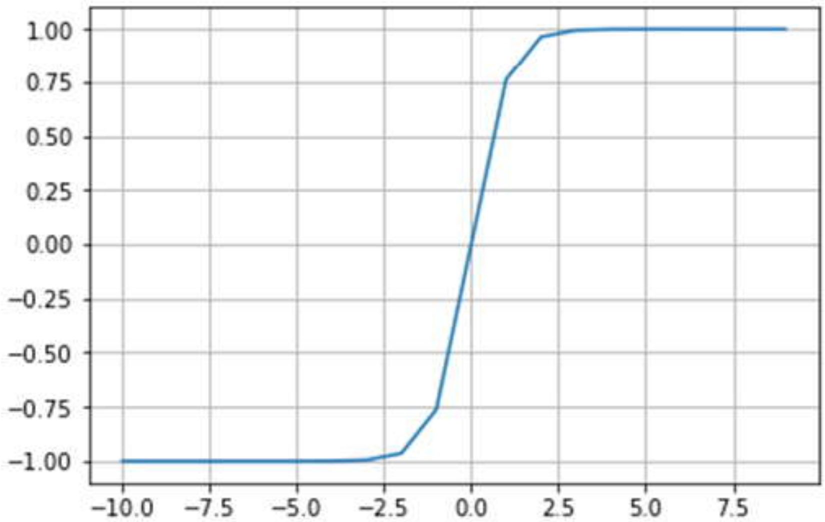
```
def tanh_array(x):
    return numpy.tanh(x)
```

Generate numbers from -10, 10 with step 1.

```
numbers = numpy.arange(-10,10)
tanh = tanh_array(numbers)
```

Generate output graph

```
fig, ax = plt.subplots()
ax.plot(numbers, tanh)
ax.grid(True)
plt.show()
```



10.2.2.3 ReLu function

ReLu stands for rectified linear units. It is given as:

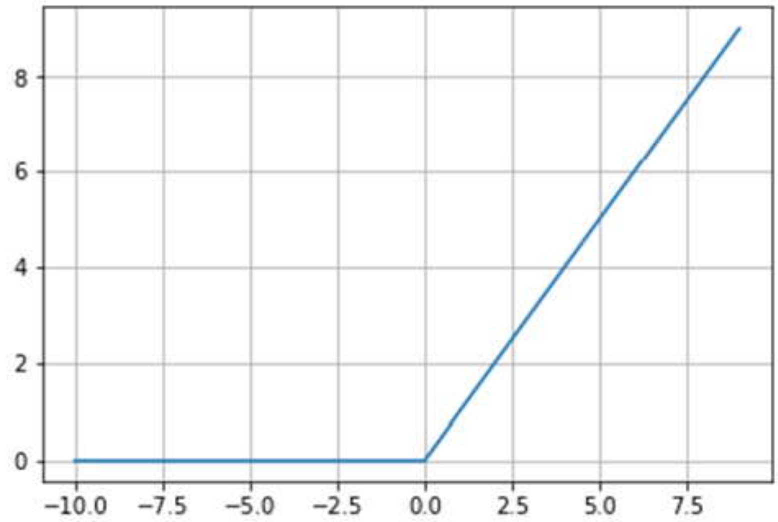
$R(x) = \text{Max}(0, x)$

So ReLu function gives output 0 for all negative value otherwise same as given number.

```
def relu_array(x):
    return numpy.maximum(0,x)
```

Generate output

```
relu = relu_array(numbers)
fig, ax = plt.subplots()
ax.plot(numbers, relu)
ax.grid(True)
plt.show()
```



ReLu is the nonlinear function. It generates output from (0,inf). So, this function will activate only those nodes which gives positive output from ReLu. By eliminating negative values, it helps in speed up the process. This is mostly used function in neural networks.

So, which function to choose is depends on the requirement. We can create our own activation functions and use them.

10.2.3 Parameter Initialization

In neural network we need to initialize parameters by some value. We can initialize b to zero but not to w. If we initialize nodes in one layer, it produces same output. So even after weight update, all nodes update by same amount. Hence, all hidden units compute the same function.

To deal with this limitation we initialize w with small random values. If we use large values, activation function (sigmoid, tanh) results vanishes gradient (only results extreme values). So, the learning slows down.

We will build our first simple neural network classifier by using keras library. For this experiment we will use iris dataset. This is a public dataset downloaded from UCI https://archive.ics.uci.edu/ml/datasets/iris

For this task we need to install two additional libraries Keras and Tensorflow. Use the following command to install keras and tensorflow.

```
! pip install keras
! pip install tensorflow
```

Import libraries needed for neural network modeling.

```
#Import libraries
import seaborn
import numpy
import pandas

from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import np_utils
```

Load data.

```
#Read iris dataset stored in data folder
data = pandas.read_csv('D:/book/data/iris.csv')
```

Check the data

```
data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Above data has three classes in text form. To give input to our neural network model, we first need to convert them to one hot encoding. We use pandas get_dummies() function to create one hot encoded class.

```
class_as_one_hot_encoding = pandas.get_dummies(data['class'])
```

Remove class and concat one hot encoding values

```
del data['class']
data = pandas.concat([data, class_as_one_hot_encoding],
axis=1)
```

Separate independent and dependent attributes

```
x = data.values[:,:4]
y = data.values[:,4:]
```

Training and test split

```
train_x, test_x, train_y, test_y = train_test_split(x, y,
train_size=0.7)
```

Keras is a library of neural network. It has a lot of hyper parameters to tune for better model creation. In the dataset which we have chosen contains 4 features and one output with three possible classes. So, our input layer must have four units and output layer must have three units. We have a choice of number of hidden layers and hidden units within.

In this example we will use only one hidden layer and take 10 hidden units in it.

We will use sequential model in keras. This model is used to stacking up layers.

```
model = Sequential()
```

Now, we define our first hidden layer. This layer is connected with four attribute input layer. In Keras we have different type of layers

- **Dense Layer:** Nodes are fully connected with the output layer nodes.
- **Activation Layer:** Contains activation functions like sigmoid, ReLu, tanh
- **Dropout Layer:** Used for regularization
- **Flatten Layer:** Flatten the input.
- **Reshape Layer:** Reshape the output.
- **Permute Layer:** Permute the dimension of input as per pattern
- **Repeat vector:** Repeat the input n times.

In our example, we will use Dense layer followed by activation layer. In our first simple neural network we want to connect each input layer node to output layer node. We will use 10 hidden units in hidden layer.

```
model.add(Dense(10, input_shape=(4,)))
```

We will use sigmoid activation for this layer.

```
model.add(Activation('sigmoid'))
```

Now we define output layer. As we have 3 output classes, we use three units in output layer.

```
model.add(Dense(3))
```

We use softmax activation function for output layer.

```
model.add(Activation('softmax'))
```

Now we configure neural network training process. We have following choices.

10.2.4 Optimizer

- Stochastic Gradient Descent
- Adamax
- Adam
- Nadam
- TFOptimizer
- RMSprop

10.2.5 Loss Function

Loss function is used to create the model for prediction. Different type of matrices are defined for different type of data. Some of the popular loss functions are:

- mean_squared_error
- mean_absolute_error
- binary-cross-entropy
- categorical_crossentropy
- poisson

Now, we compile our model by using compile() function. To compile the model, we use efficient library (Tensorflow in this case) to do the processing.

```
model.compile(optimizer='adam', loss='categorical_
crossentropy', metrics=["accuracy"])
```

Now, we fit the model. We can choose different set of hyper parameters which affects our training. Here we have chosen number of iteration = 100 (epochs) and batch size =1. Batch size = 1 means at one time we are exposing only one training example to our model. If we don't want to see the processing and loss calculation, we can add parameter verbose=0 in the fit function.

```
model_info = model.fit(train_x, train_y, epochs=100,
batch_size=1 )
```

Output:

Epoch 1/100

105/105 [==============================] - 0s - loss: 1.2940 - acc: 0.3333

Epoch 2/100

105/105 [==============================] - 0s - loss: 1.1401 - acc: 0.3333
Epoch 3/100
105/105 [==============================] - 0s - loss: 1.0605 - acc: 0.3333
Epoch 4/100
105/105 [==============================] - 0s - loss: 0.9929 - acc: 0.4190
Epoch 5/100
105/105 [==============================] - 0s - loss: 0.9046 - acc: 0.6667
.
.
.
Epoch 98/100
105/105 [==============================] - 0s - loss: 0.1014 - acc: 0.9905
Epoch 99/100
105/105 [==============================] - 0s - loss: 0.1001 - acc: 0.9905
Epoch 100/100
105/105 [==============================] - 0s - loss: 0.1010 - acc: 0.9905

We use keras evaluate() function which returns loss and accuracy

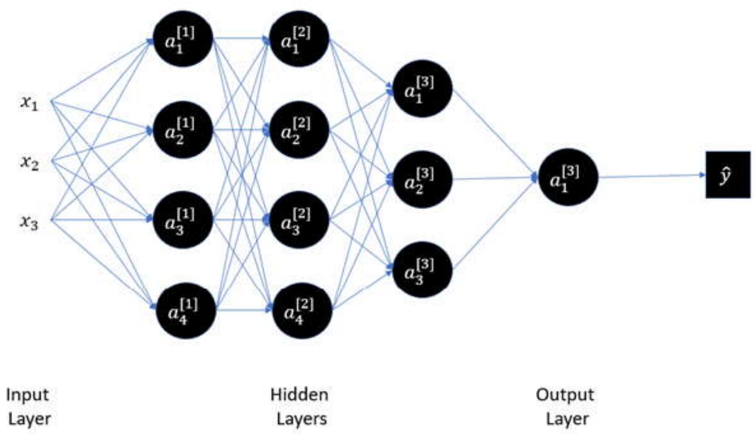
```
loss, accuracy = model.evaluate(test_x, test_y, verbose=0)
print("Model Accuracy = {:.4f}".format(accuracy))
```

Output:

Model Accuracy = **0.9333**

10.3 Deep Learning

Deep learning is nothing but a neural network which contains large number of layers.



Above is 4-layer neural network. It contains 3 hidden layers and one output layer. Number of units in any layer is denoted by $n^{[l]}$. Hence $n^{[1]} = 4$, $n^{[2]} = 4$, $n^{[3]} = 3$, $n^{[4]} = 1$, As number of hidden units in first layer is 4, second layer is 4, third layer is 3 and forth layer is 1. $a^{[l]}$ used to denote activation at layer l.

Using deep network, we can compute function with fewer number of hidden units.

Deep learning rise happens because of the computational power increment. Earlier models are trained on CPUs which have 4-24 cores. These cores can be used to do parallel processing and do computations.

In Neural Network, we have to do computations on each node and collect the result. As with limited cores, our parallel processing bounds to the number of cores available.

In GPUs, 1000-4000 processing cores are available to do the parallel computation. Deep learning hidden unit calculation can be done in these cores and results are multiple times faster.

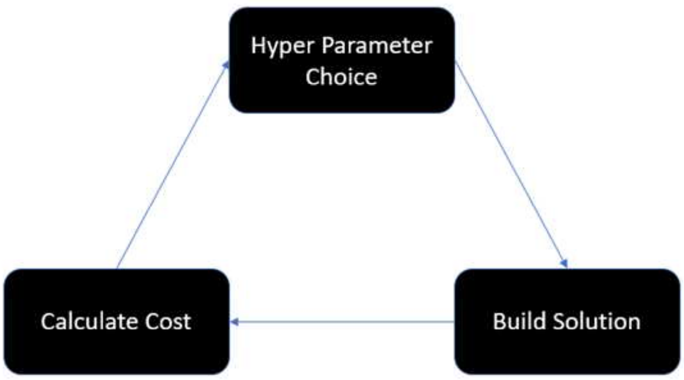
10.3.1 Hyper parameters

Deep learning model has a lot of choice for hyper parameters

- Learning Rate
- Number of hidden layer
- Number of hidden units
- Number of iteration to get the minima
- Activation function
- Momentum
- Batch size
- Regularization term

These parameters controls w and b. These are called hyper parameters.

So, Deep learning solution works in this way:



First, we start with choice of hyper parameters. Then we build the solution by

using programming language frameworks. Finally, we calculate cost for the solution. We can choose different set of hyper parameters and repeat this cycle.

10.4 Deep Learning Architecture

With the advancement in computing power and rise of GPUs, different variants of neural networks are available to solve problems in different domains. In this section, we will learn about some popular deep learning architecture

- Deep Belief Network (DBN)
- Convolution Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Long Short-Term Memory (LSTM)
- Deep Stacking Network (DSN)

Let's discuss each architecture one by one.

10.4.1 Deep belief networks

Deep Belief Network (DBF) is multilayer network. Each two layers are restricted Boltzmann machines (RBM). We can assume DBN as a stack of RBMs.

In DBN each hidden layer understands abstract representation of input feature. Output layer is different. It performs classification task. In DBN two tasks are performed - Unsupervised pretraining and supervised fine-tuning.

Unsupervised pretraining: in this RBM is trained to reconstruct its own input. Each layer takes input from previous layer.

Supervised fine-tuning: in this, labels are applied and gradient descent with back propagation algorithm is used to do training.

10.4.2 Convolutional neural networks

Convolutional neural networks (CNN) are based on visual cortex. It assumes that input are images and extract properties in different layers. It's multiple hidden layer do feature extraction followed by classification. Feature extraction task divide in convolution and pooling. First image is fed to convolution layer and features are extracted. After this pooling is done to reduce the dimensionality of extracted feature. Again, on these reduced dimensions, convolution and pooling is done. Multiple layers of convolution and pooling is done which extract feature at different levels. Final, layer outputs the overall features of images.

10.4.3 Recurrent neural networks

Recurrent neural network has connection that feed back to past layer or in the same layer. This implementation allows network to maintain memory of past and current events. RNN is trained by using a variant of back propagation which is called back propagation through time. In this technique weight updates are done by summing all the weight updates of errors of elements which comes in the sequence.

RNN can store information while processing input. This kind of architecture is very

useful when past information helps in prediction of future information (like in case of time series analysis).

RNN maintains context nodes to store the information. Hidden layer output is applied back to hidden layer but whole network remains feed forward.

10.4.4 Long Short-Term Memory

Long Short Term Memory (LSTM) networks are based on memory cell. Memory cell can store the information for long time or short time depending on the importance of information.

Each LSTM memory cell contains three types of gate, input gate, forget gate and output gate.

- Input gate allows information to enter in the cell.
- Forget cell allows information to be forgotten or removed from the cell. By this useless information is removed and space is created for any new useful information.
- Output gate is used to output information from the cell.

10.4.5 Deep Stacking Network

Deep stacking network is a set of deep networks. Each network has its own set of hidden layers.

In one instance of DSN, three modules are present. Each module contains input layer, one hidden layer and output layer. These three modules are stacked on each other. One module input contains output of previous module.

10.5 Deep Learning Framework

Today with the power of deep learning to solve complex problem, a lot of frameworks exists. Using these framework, applying deep learning in different problems is very easy.

Tensorflow

It was developed by google brain team and open sourced in 2015. It is a python based library which can run on multiple CPUs and GPUs.

Theano

It is one of the early deep learning library. It lacks supports for multiple GPUs. It is good for numerical computation on CPUs and GPUs.

Keras

Keras is built over theano and tensorflow. It uses them to build models. As building model in above two frameworks is a challenge, keras does the work to layer it and present easy way to users to create models. We also used this library for our deep learning program implementation. It is easy to learn, has good documentation and support.

Torch

It was developed by facebook, google and twitter. The aim of this library is to make the process of building deep learning model simple.

Deeplearning4j

It is deep learning framework to support Java and other JVM supported languages.

Caffe

It is one of the oldest deep learning library. It was made to do deep learning in C++ but it also has implementation in python. It is used majorly to design convolution neural networks.

Now we will see some complex problem where deep learning can be used to improve performance.

Object recognition is the process to identify predefined objects in the images. In the following example, we will use Keras to create CNN for object recognition.

We will use CIFAR-10 dataset in this task. CIFAR stands for Canadian Institute for Advanced Research (CIFAR). This dataset has 60000 images of different objects.

This dataset can easily be loaded by using keras library. When we execute the code, it downloads the images. It may take some time depending on the bandwidth.

Import the libraries

```
from keras.datasets import cifar10
from keras.layers import Dropout
from keras.layers.convolutional import MaxPooling2D
from keras.layers import Flatten
from keras.constraints import maxnorm
from scipy.misc import toimage
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras import backend as K
K.set_image_dim_ordering('th')
```

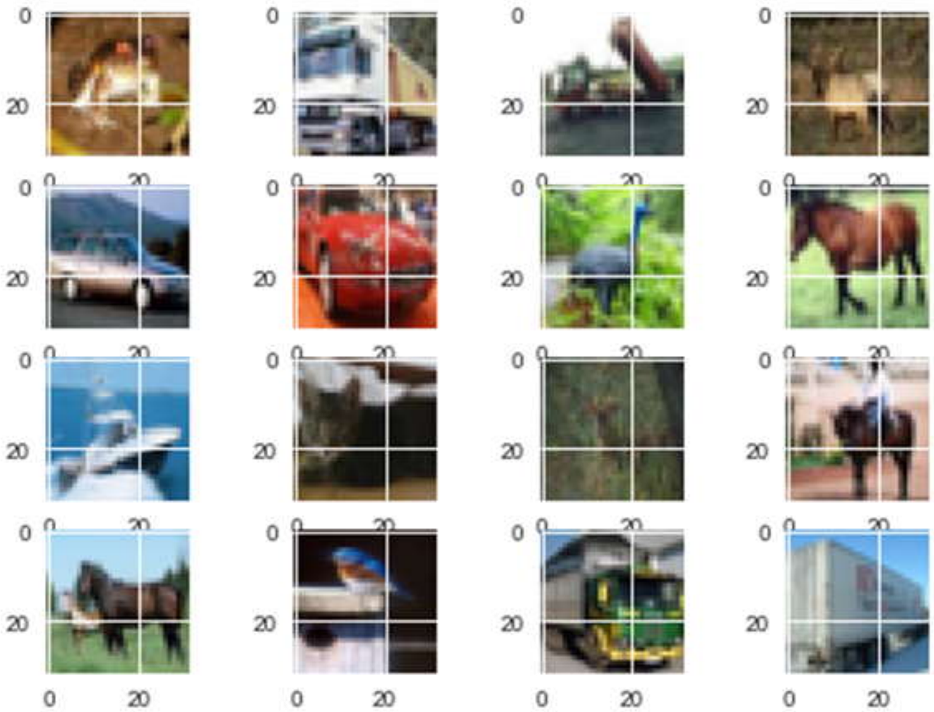
Loading the data

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Let's see some examples from this data

```
fig = plt.figure()
gs = gridspec.GridSpec(4, 4, wspace=0.0)
```

```
ax = [plt.subplot(gs[i]) for i in range(4*4)]
for i in range(16):
 ax[i].imshow(toimage(x_train[i]))
plt.show()
```



Now convert the class to one hot encoding matrix

```
y_train_onehot = np_utils.to_categorical(y_train)
y_test_onehot = np_utils.to_categorical(y_test)
```

We can use simple CNN architecture to start. In this example we will use two convolutional layers and a max pooling.

```
# Create the model
#Sequential model is selected to get a stack of layers.
num_classes = 10
model = Sequential()

#First convolution layer
model.add(Conv2D(32, (3, 3), padding='same',input_shape=(3,
32, 32), activation='relu'))

#Second convolution layer
model.add(Conv2D(32, (3, 3),padding='same', activation='relu',
))
```

```
#Pooling
model.add(MaxPooling2D(pool_size=(2, 2)))

#Flatten the output
model.add(Flatten())
model.add(Dense(512, activation='relu'))

#Output class
model.add(Dense(num_classes, activation='softmax'))
# Compile model
epochs = 50
lrate = 0.05

sgd = SGD(lr=lrate, momentum=0.8, decay=lrate/epochs,
nesterov=False)

#Compile the model
model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['accuracy'])

#Print summary of CNN
print(model.summary())
```

Output:

```
_________________________________________________________________
Layer (type)                  Output Shape           Param #
=================================================================
conv2d_1 (Conv2D)             (None, 32, 32, 32)     896
_________________________________________________________________
conv2d_2 (Conv2D)             (None, 32, 32, 32)     9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2  (None, 32, 16, 16)     0
_________________________________________________________________
flatten_1 (Flatten)           (None, 8192)           0
_________________________________________________________________
dense_3 (Dense)               (None, 512)            4194816
_________________________________________________________________
dense_4 (Dense)               (None, 10)             5130
=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
_________________________________________________________________
None
```

Fitting the model (It will take time because the process is heavy)

```
model.fit(x_train, y_train_onehot, validation_data=(x_test,
y_test_onehot), epochs=250, batch_size=100)
```

Output:

Train on 50000 samples, validate on 10000 samples

Epoch 1/250

50000/50000 [==============================] - 489s - loss: 14.5072 - acc: 0.0999 - val_loss: 14.5063 - val_acc: 0.1000

Epoch 2/250

4000/50000 [=>............................] - ETA: 409s - loss: 14.5023 - acc: 0.1002

Final evaluation of the model

```
loss,accuracy = model.evaluate(x_test, y_test_onehot,
verbose=0)
print("Model Accuracy = {:.4f}".format(accuracy))
```

Model Accuracy = **0.9876**

Chapter 11

Recommendation System

Recommender Systems are applications or platforms that recommend useful things to the user according to the context. The goal of recommendation systems are to present items to user which are interesting to them. It helps in selection and increase in sales. In our day to day life, we find ourselves surrounded by recommender systems. While we search for product on any popular e-Commerce website, it starts recommending similar products. For example, if we search for camera and buy it, next time system starts recommending camera stand, cover etc. In video sharing platform, like youtube, when we watch any video, it starts recommending similar videos. In movie rating website like IMDB, when we rate or search for any movie, it starts recommending similar movies.

Some popular recommendation systems are

- **Product Recommendation:** Amazon, Snapdeal, Walmart
- **Music Recommendation:** Last.fm, youtube
- **Movie Recommendation:** Jinni, Movielens

This chapter deals with strategy and ways use to develop recommender engines. We will learn about the following techniques of building recommendation engines in this chapter:

1. Popularity Based
2. Content Based
3. Classification Based
4. Collaborative Filtering
5. Model Based

Let's discuss these strategies one by one.

11.1 Popularity Based Recommender Engines

Popularity based recommendation engines are based on the usage of items by all the users. The items which is used most is consider as most popular item. These systems recommend popular items to the all the users.

These systems are based on the past usage count of all items. Highest count item is recommended to all the users.

These kinds of recommendation systems are used in news articles which recommend top items to each new user. If any story is clicked by more number of users, its count goes higher, and it is recommended to all the users.

One drawback of these systems is that they assume each user as same type. They recommend same items to each user irrespective of their profile, geography and other attributes. The reason for this is they don't consider user data.

In this task we will use books data. This dataset is downloaded from: http://www2.informatik.uni-freiburg.de/~cziegler/BX/ Collected by Cai-Nicolas Ziegler in a 4-week crawl (August / September 2004) from the Book-Crossing community with kind permission from Ron Hornbaker, CTO of Humankind Systems. Contains 278,858 users (anonymized but with demographic information) providing 1,149,780 ratings (explicit / implicit) about 271,379 books.

First to import libraries we need to build the model:

```
import pandas
import numpy
```

We have modified the original dataset slightly. You can use this from data/books_recommendations folder. It contains Books, Users and Ratings table.

```
#As data contains non alpha numeric characters, we used
latin1 encoding while loading the data
books = pandas.read_csv('./data/books_recommendation/
Books.csv',encoding='latin1')
users = pandas.read_csv('./data/books_recommendation/
Users.csv',sep=';',encoding='latin1')
books_ratings = pandas.read_csv('./data/books_recommendation/
Book-Ratings.csv',encoding='latin1')
```

We can check the sample values of the tables by head() function.

```
books.head()
```

books_rating table contains ratings of the books whose information is not available in books table. First get the list of ISBN and remove entries whose books details are not present.

```
#Check shape of each dataframe
print('Books : ',books.shape)
print('Users : ',users.shape)
print('Books Ratings : ',books_ratings.shape)
```

Get all ISBN from books table

```
isbn = books.ISBN.unique()
```

Take the books ratings

```
books_ratings =
books_ratings[books_ratings['ISBN'].isin(isbn)]
```

Check the size of book_ratings dataframe

```
print('Books Ratings : ',books_ratings.shape)
```

Group by ISBN

```
isbn_count_list = books_ratings.groupby('ISBN')['Book-
Rating'].count()
isbn_count_frame = pandas.DataFrame(isbn_count_list)
```

Get the list of top-15 books

```
top_15_isbn = top_15_isbn.index.values
```

Create a dataframe from top isbn

```
top_books = pandas.DataFrame(top_15_isbn,columns=['ISBN'])
```

Fill the other books details

```
top_books = pandas.merge(top_books,books,on='ISBN')
```

Get details of top books

```
top_books = top_books[['ISBN','Book-Title','Book-Author',
'Publisher']]
```

Show top books

```
top_books
```

	ISBN	Book-Title	Book-Author	Publisher
0	0971880107	Wild Animus	Rich Shapero	Too Far
1	0316666343	The Lovely Bones: A Novel	Alice Sebold	Little Brown
2	0385504209	The Da Vinci Code	Dan Brown	Doubleday
3	0060928336	Divine Secrets of the Ya-Ya Sisterhood: A Novel	Rebecca Wells	Perennial
4	0312195516	The Red Tent (Bestselling Backlist)	Anita Diamant	Picador USA
5	044023722X	A Painted House	John Grisham	Dell Publishing Company
6	0142001740	The Secret Life of Bees	Sue Monk Kidd	Penguin Books
7	067976402X	Snow Falling on Cedars	David Guterson	Vintage Books USA
8	0671027360	Angels & Demons	Dan Brown	Pocket Star
9	0446672211	Where the Heart Is (Oprah's Book Club (Paperba...	Billie Letts	Warner Books
10	059035342X	Harry Potter and the Sorcerer's Stone (Harry P...	J. K. Rowling	Arthur A. Levine Books
11	0316601950	The Pilot's Wife : A Novel	Anita Shreve	Back Bay Books
12	0375727345	House of Sand and Fog	Andre Dubus III	Vintage Books
13	044021145X	The Firm	John Grisham	Bantam Dell Publishing Group
14	0452282152	Girl with a Pearl Earring	Tracy Chevalier	Plume Books

Check total number of authors

```
books['Book-Author'].describe()
```

We can see that total 102,042 unique authors are present. In top 15, John Grisham and Dan Brown appeared twice.

11.2 Content Based Recommendation Engine

These recommendation engines are based on features of the items. Items are compared or recommended based on their feature values. Item having similar properties gets higher weights.

We will show an example by using nearest neighbor algorithm. Nearest neighbor algorithm is an unsupervised learning algorithm. It is known as **memory-based system** because it memorizes the items and for any new instances or item recommend the similar item from its memory. So, if the training data is large and cannot be fitted in memory, then these systems are not preferred.

Let's take an example of online mobile shop. It contains different types of mobile with different features. Consider this mobile database

Mobile-ID	Battery Life (In Hrs.)	Camera (In MP)	ROM (In GB)
ACT-23S	48	12	16
ACT-56P	42	20	64
NTK9	60	8	32
LK0	72	4	16
LK3	48	16	32

Now a user started searching for different product. Website first show him top popular mobiles chosen by other users. Now, user decided to put a filter and see the mobile which matches the criteria most.

User has the following requirement

Battery Life (In Hrs.)	Camera (In MP)	ROM (In GB)
55	10	32

Based on user's requirement most suitable mobile is NTK9. Let's understand this concept by using python code.

We will use house dataset. This dataset is created from user responses. This is available in /data folder. This dataset contains following fields

- **ID:** Unique ID of each property
- **Size of House:** Small, Medium, Large, Big
- **Number Of Bathrooms:** Number of Bathrooms available in the house
- **Age of building:** Less than a year, 1-5 Years, 5-10 Years, 10+ Years

- **Number of Fireplace:** 0,1
- **Furnished Status:** Unfurnished, Semi Furnished, Fully Furnished
- **Car Parking:** Yes/No
- **Air Conditioning:** Yes/No
- **Private Garden:** Yes/No
- **Life Available:** Yes/No
- **Area Status:** High, Medium, Low
- **Rent:** Integer

Import libraries that is needed for this recommendation engine.

```
import pandas
import numpy
import sklearn
from sklearn.neighbors import NearestNeighbors
```

Read Survey data

```
house_data = pandas.read_csv('./data/House_Survey.csv')
```

Check initial data

```
house_data.head()
```

To apply any algorithm first we will convert categorical variable to numeric numbers. We can do this with predefined functions in python. But we want to assign weights according to their categories.

```
#Convert Size
house_data.loc[house_data['Size of House'] == 'Small', 'Size
of House'] = 1
house_data.loc[house_data['Size of House'] == 'Medium', 'Size
of House'] = 2
house_data.loc[house_data['Size of House'] == 'Large', 'Size
of House'] = 3
house_data.loc[house_data['Size of House'] == 'Big', 'Size
of House'] = 4
```

We do similar for all the other attributes.

Now, we start by creating nearest neighbor object. We use this algorithm to get the data point which is nearest to the requirement (neighbour =1).

```
nn1 = NearestNeighbors(n_neighbors=1)
```

Fit model

```
nn1.fit(house_data.loc[:,'Size of House':'Area Status'])
```

Now define our requirement in the same format (not taking ID and rent values).

```
requirement = [2,2,3,1,0,2,1,0,0,0,1]
```

Check the most suitable house for this requirement

```
print(nn1.kneighbors(requirement))
```

Output:

```
(array([[ 1.73205081]]), array([[5]], dtype=int64))
```

Checking the value in house_data at index = 5.

```
house_data.loc[[5]]
```

If we want to get top-3 list

```
top3 =
nn1.kneighbors(requirement,n_neighbors=3)[1].tolist()[0]
```

Check top3 house of the test house

```
house_data.loc[top3]
```

11.3 Classification Based Recommendation Engine

In this technique we can use classification methods to recommend items. We have seen different classification techniques (Logistic regression, decision tree, random forest, naïve bayes) in previous chapters. We will build our model by using logistic regression in this section.

The advantage of these recommendation is personalization. They can take data of users, items, user's past purchase history and some other contextual data if present. Based on value of different parameters, model can decide that user will purchase this product or not.

We can enhance the power of these recommendations with rich contextual data. For example, on an online mobile store, contextual data can be hour of the day, season, festival season or not, day of the month, cookies of the user's browser etc.

Consider another example of car website. The website provides free test drive at home to selected users. To participate in this offer, user must fill a form and provides correct details. Suppose these are the fields which users need to answer.

House Location
Employment Status
Previous Car
Number of Active loans
Number of Credit cards
Loan defaulter status
Single or Married
Gender
Family Size

Now, we can build a classifier that classify each user in two categories, which are: Potential buyer or not. If the customer looks like a potential buyer, information should be forwarded to marketing team and they can proceed with the offer. If the customer does not seem to be a potential buyer, there is no point to give him an offer of free home test ride. So, a machine learning classifier seems to be most appropriate in this case. It can use past customer data to build classifier.

To understand classification-based recommendation we will use bank marketing data. This dataset is available publicly and can be downloaded from: http://archive.ics.uci.edu/ml/datasets/Bank+Marketing. This dataset is based on marketing calls of Portuguese banking institution.

Import additional libraries

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE
```

Read the bank data in pandas dataframe

```
bank_data = pandas.read_csv('.\\data\\bank_data.csv',sep=';')
```

Show all columns

```
pandas.set_option('display.max_columns', None)
```

Check the head of the dataframe

```
bank_data.head()
```

In this task we will use only few attribute to build our first classification model.

```
bank_data =
bank_data[['age','job','marital', 'education', 'default',
'housing', 'loan','contact','y']]
```

Convert categories to 0 and 1

```
bank_data.y.replace(('yes', 'no'), (1, 0), inplace=True)
```

Check the shape of the data

```
bank_data.shape
```

Extract features form the dataset

```
X = bank_data.iloc[:, :7]
X = pandas.get_dummies(bank_data.iloc[:, :7]).values
```

Extract class

```
Y = bank_data.iloc[:, 8:].values
```

Split the data in training and test set

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.10)
```

Create object of logistic regression

```
logistic_regression = LogisticRegression()
```

Over sampling using SMOTE algorithm

```
smote_object = SMOTE(random_state=4)
X_res, y_res = smote_object.fit_sample(X_train, Y_train)
```

Fit the model

```
logistic_regression.fit(X_res,y_res)
```

Predict using logistic regression model

```
Y_pred = logistic_regression.predict(X_test)
```

Calculate Accuracy

```
accuracy_score(Y_test,Y_pred)
```

Accuracy of this model is less. But this was only the demonstration to use classification in recommenders. We can use other improvement like more features, feature engineering, better algorithms, parameter tuning to get better results.

11.4 Collaborative Filtering

Collaborative filtering is based on the idea of similar interest. Take an example of two users Ankita and Neha. Ankita likes product 1,3,4 and Neha likes products 3,4,7. As both are agreed on two products, it is highly that Ankita will like product

7 and Neha will like product 1. This technique is completely based on past behavior of users. This is most widely used technique in recommendation engine building.

There are two types of collaborative filtering exists:

- **User-User collaborative filtering:** based on similar users, based on their item purchase.
- **Item-Item collaborative filtering:** based on similar items, based on purchased by users.

In this section we will focus on item-item collaborative filtering. We will use Pearson correlation coefficient to recommend item to the user based on his past item selection. Correlation between items is calculated to understand their similarity based on user reviews.

Pearson correlation coefficient is the measure of correlation between two variables. It is represented by r.

r = 1: Strong positive correlation

r = –1: Strong negative correlation

r = 0: No correlation

To demonstrate collaborative filtering we will use MovieLens dataset provided by grouplens for research. This dataset can be downloaded from: https://grouplens.org/datasets/movielens/.

```
movies = pandas.read_csv('./data/grouplens_movies.csv')
movies_ratings = pandas.read_csv('./data/
grouplens_ratings.csv')
```

Check the head of the dataframe.

```
movies.head()
movies_ratings.head()
```

First, we check the mean rating given to each movie.

```
movies_ratings_Data =
pandas.DataFrame(movies_ratings.groupby('movieId')['rating'].mean())
movies_ratings_Data.head()
```

Let's see how popular each movie is among the users. We can check this by adding one more column count to our movie_rating_Data DataFrame.

```
movies_ratings_Data['Count'] =
pandas.DataFrame(movies_ratings.groupby('movieId')['rating'].count())
movies_ratings_Data.head()
```

Now, we have to build user-item utility matrix. We can do this by using pivot_table() function of pandas.

```
movies_crosstab = pandas.pivot_table(data=movies_ratings,
values='rating', index='userId', columns='movieId')
movies_crosstab.head()
```

Above crosstab is full of null values. This is because each user rated very less movies. Hence, the above matrix is sparse.

```
#Let's take an example of movie with movieid=260
movies.loc[movies['movieId'] == 260]
```

This movie is Star Wars IV. Create a variable and remove null values.

```
star_wars = movies_crosstab[260]
star_wars = star_wars[star_wars>=0]
star_wars
```

Now our objective is to find similar movies. We will understand this by finding movies similar to Star Wars.

```
similar = movies_crosstab.corrwith(star_wars)
similar_dframe =
pandas.DataFrame(similar,columns=['PearsonR'])
```

Drop NA values

```
similar_dframe.dropna(inplace=True)
similar_dframe.head()
```

Above code shows each movie id and their correlation with star_wars. This is not very useful at this moment.

We also want to check for popular movies only which are rated by atleast 50 users. To do this let's join our frame with rating.

```
similar_dframe_summary =
similar_dframe.join(movies_ratings_Data['Count'])
similar_dframe_summary =
similar_dframe_summary[similar_dframe_summary['Count']>=50].
sort_values('PearsonR', ascending=False).head(5)
```

Get top 5 similar movies

```
top_5_similar =
similar_dframe_summary.head(5).index.values.tolist()
```

Print top 10 similar movies to Star Wars

```
movies.loc[movies['movieId'].isin(top_5_similar)]
```

Output:

	movieId	title	genres
232	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
953	1196	Star Wars: Episode V - The Empire Strikes Back...	Action|Adventure|Sci-Fi
966	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventure|Sci-Fi
6944	59315	Iron Man (2008)	Action|Adventure|Sci-Fi
7218	68358	Star Trek (2009)	Action|Adventure|Sci-Fi|IMAX

Above result is very interesting. First record is the movie itself. But second and third are the next part of similar movies. So, if a user likes Star Wars IV, we can recommend him another two movies in the same series. These results are self-explainable. This is also a reason why collaborative filtering method are very popular in recommendation engine techniques.

Chapter 12

Time Series Analysis

Time series contains a sequence of data points which are indexed in an order. This order is basically the time in which they happened or recorded. Points are recorded at discreate time with equal space intervals. Some of the examples are temperature over the year, stock price sequence at every five-minute interval, visitors on a website during holiday week etc.

Time Series analysis includes methods to analyze the time series data points and make meaningful insights. These insights can be used as business perspective to get the trends and direction in past.

Time series forecasting is the way to predict the behavior by using the past data points. Forecasting is very much useful to know the direction of data. For example, consider the demand for a drug from a pharmaceutical company. They can create points for there sales in past and predict demand in future. That is very useful to fulfill the drug requirement and do not waste any additional resource.

Time series forecasting is different from regular supervised learning. In supervised learning we assume that all the observations are independent from each other. Data recorded can be used in any order and that do not have any sequence to follow. This is not the case with time series. In time series, data points sequence makes a lot of sense and we cannot ignore it. Also, with sequence, we can find short term trends in data points. For example, sales of drug may increase in June due to bad weather conditions. After the time frame, demand goes down.

In this chapter we will learn about time series, how to analyze and different method of forecasting.

12.1 Date and Time Handling

Let's start with date and time related functions available in python to deal with time series. Pandas support many date and time functions. We will review some of them here.

First import the library to run the code of this module.

```
import pandas
import numpy
```

Using pandas we can generate data in the range of time. Consider the date_range functions.

```
pandas.date_range('2018Jan 16', periods = 20, freq = 'W')
```

Above function generates the data for 20 days starting from Jan 16, 2018 and frequency is weekly. First parameter is the starting date. period is the count we want to generate. freq stands for type of frequency in the sequence.

```
pandas.date_range('2018 Jan 16', periods = 20, freq = 'D')
```

Above function generate data for 20 days starting from 16 Jan 2018 and frequency is daily. We can generate data in timely manner also.

We can also specify timezone with date_range function in pandas.

```
pandas.date_range('2018 Jan 16', periods = 10, freq = 'D',
tz='America/New_York')
```

Above will create the date range with specific timezone consideration.

Timestamps are used to denote one moment of time. It represents time in discreate levels. pandas have timestamp related function which help us to do that.

```
pandas.Timestamp('2018 Jan 16 22:33:44')
```

Here, we want to capture the moment or time flag at the specified time. pandas timestamp can go up to Nano second. That is useful when we are working with a data generating continuously from large industrial machines (e.g. Aircraft data, wind turbine data etc.)

Timestamp has a lot of properties. Some of the useful properties are:

Property	Description
second	Second of the timestamp
minute	Minute of the timestamp
hour	Hour of the timestamp
day	Day of the timestamp
week	Week of the timestamp
month	Month of the timestamp
year	Year of the timestamp
dayofweek	Day of the week of the timestamp
weekday_name	Day name of a week
is_leap_year	Year is leap year or not

We can check examples here

```
first_stamp = pandas.Timestamp('2018 Jan 16 22:33:44')
print('Day of the timestamp',first_stamp.day)
print('Week of the timestamp',first_stamp.week)
```

```
print('Minute of the timestamp',first_stamp.minute)
print('Day of the week of the timestamp',first_stamp.dayofweek)
print('Year is leap year or not',first_stamp.is_leap_year)
```

We can add time in timestamp and move the timestamp forward or backward in time. This can be done by using Timedelta function in pandas.

```
first_stamp + pandas.Timedelta('1 day')
```

We can add hours, minutes or seconds in the same way.

```
first_stamp + pandas.Timedelta('1 hour 10 minutes')
```

Also, we can move backward and substract the time

```
first_stamp - pandas.Timedelta('50 seconds')
```

Timestamp represent one moment of time. When we want time range, we consider Period.

```
period = pandas.Period('Jan 2016')
print('Start time', period.start_time)
print('End time',period.end_time)
```

We can use this to check the validity of timestamp within the period or not

```
print(period.start_time<first_stamp and
period.end_time>first_stamp)
```

We can also create buckets of periods within a range.

```
pandas.period_range('2018-01-16','2018-01-30', freq = 'D')
```

In above function, first parameter is the start time, second is the end time and freq denotes what should be the range of one period. Here, we are specifying frequency as day.

We can also use timezone related functionality along with Timestamps

```
second_stamp = pandas.Timestamp('2018 Jul 6 10:10:10')
second_stamp.tz_localize('Asia/Kolkata')
```

Above function gives localize timestamp according to 'Asia/Kolkata'.

We can localize the same timestamp again.

```
third_stamp = second_stamp.tz_localize('Africa/Addis_Ababa')
third_stamp
```

We can convert one timezone Timestamp to other timezone. Timestamp time and date will change accordingly.

```
third_stamp.tz_convert('America/Manaus')
```

tz_localize() function change the timezone but do not convert it. tz_convert() will convert the timestamp according to new timezone. This is very useful when we have data from different timezones and we want to make it unique and arrange in the sequence.

We can check all the available timezone in pandas.

```
from pytz import all_timezones
print(all_timezones[1:20])
```

Let's talk about reading date data in python. We can read the file in python by using file open

```
with open('.\data\monthly_price.csv') as f:
    for x in range(10):
        print(next(f))
```

We can use pandas read_csv function to read the file in pandas

```
data = pandas.read_csv('.\data\monthly_price.csv')
```

We can analyze the above data. In this data, date is given as Date column. When we observe the data, it contains one entry for each month for 6 years. So, first we need to get rid of day from date.

First take year and month from the date

```
data['Year'] = data.Date.dt.year
data['Month'] = data.Date.dt.month
```

Now, change the date column

```
data['Date'] = data['Date'].astype(str)
for index,row in data.iterrows():
    data.set_value(index,'Date',str(row['Year']) + '-' +
    str(row['Month']))
```

Delete Year and Month as those are not needed anymore

```
del data['Year']
del data['Month']
```

Above operation can be consolidated and performed by using single operation.

```
data.Date = data.Date.dt.to_period('m')
```

This is called resampling of the data. Sometimes working with the date data is not in the correct desired format. In pandas we have a lot of function to resample the data according to the need.

To understand resampling let's generate some data first

```
first_series = pandas.DataFrame({'Date' : pandas.date_range
('12/15/2018', periods = 20, freq = 'H'), 'Count':list
(range(20))},columns=['Date','Count'])
```

Above code will generate a dataframe with 20 timestamps starting from 15 Dec 2017 00:00:00 and hourly frequency.

Set the Date column as index.

```
first_series = first_series.set_index('Date')
```

Now, create another series with data interval is 40 Min instead of 1 hour.

```
second_series = first_series.asfreq('40Min', method = 'ffill')
```

In above code we have chosen forward fill method. This method fills the data points forwardly. It adds the information in the new data rows with the immediate information ahead.

We can also fill the data backwardly.

```
third_series = first_series.asfreq('40Min', method = 'bfill')
```

If we don't want to add any data in new rows we can choose None in method parameter.

```
forth_series = first_series.asfreq('40Min', method = None)
```

When we add data rows, dataframe size increases. Our original dataframe has 20 rows. We can check the shape of new dataframe.

```
forth_series.shape
```

In above codes, we are adding data points in the time series data. What if we want to remove the data points. We can do this by using following

```
fifth_series = first_series.asfreq('5H', method = None)
```

Using above code, we are losing the data points. Count represent value at that point. It is not which we want always. If we are using some other statistics like sum, mean, max, min we have to use resample function.

```
first_series.resample('5H').mean()
first_series.resample('5H').sum()
first_series.resample('5H').min()
first_series.resample('5H').max()
```

Resampling needs a lot of domain knowledge. Backward or forward are completely depend on the analysis.

12.2 Window Functions

In window function, we use windows of data to do the analysis. Basically, there are two type of window functions which are defined, and these are Rolling window and expending window.

In rolling window, we take the data points and give the combined result at some point of time. We can consider it as moving summation. We can choose different size of rolling window.

Expending window keep taking the data points. At any point it will give you information considering all the past points. We can consider it as cumulative summation.

Let's start by rolling window.

```
rolling_window_5 = data.rolling(window = 5)
```

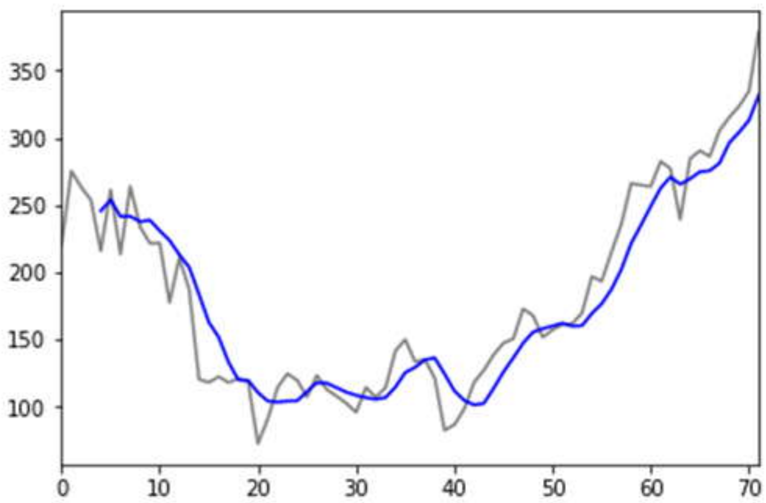
Here, we defined rolling window of size 5. It will start when it has minimum 5 data points. We can check this by printing the object

```
rolling_window_5
```

Now, create a window of mean of values and plot it

```
data['Price'].plot(color = 'gray')
rolling_window_5.mean()['Price'].plot(color = 'blue')
```

This will generate a plot like this:

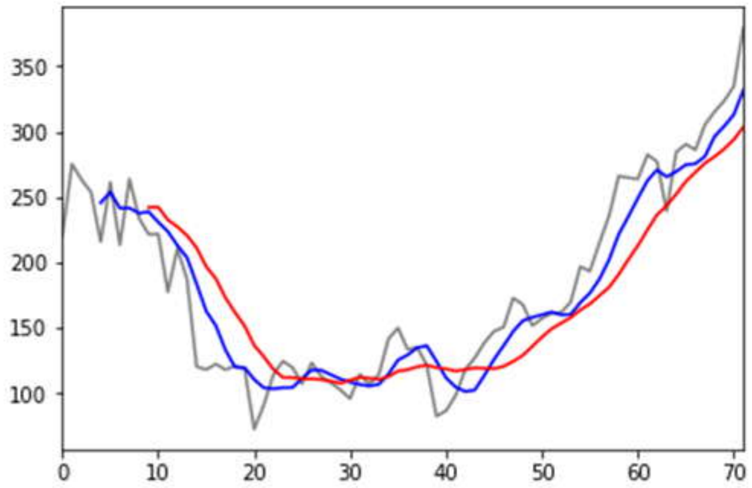


Grey line represents the data. Blue line represents the window of size 5.

To get more smooth result, we can use window of greater size.

```
rolling_window_10 = data.rolling(window = 10)
data['Price'].plot(color = 'gray')
rolling_window_5.mean()['Price'].plot(color = 'blue')
rolling_window_10.mean()['Price'].plot(color = 'red')
```

This will generate a plot like this:

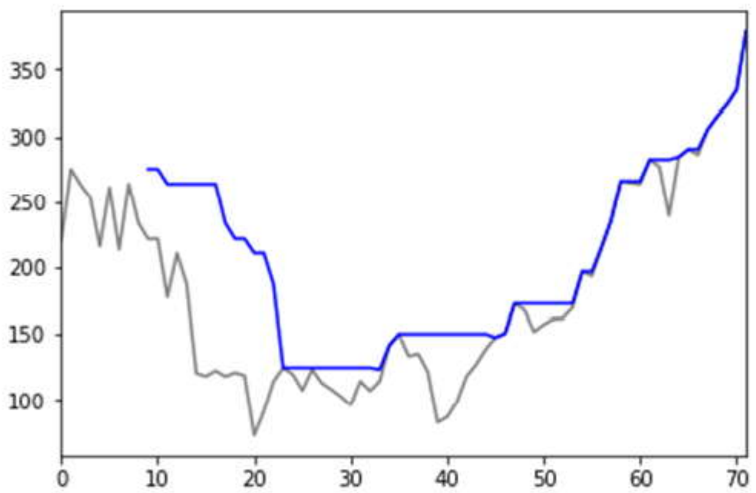


Here, red curve represents a smoother version with greater window size.

We can use different kind of function instead of mean. For example, max function.

```
data['Price'].plot(color = 'gray')
rolling_window_10.max()['Price'].plot(color = 'blue')
```

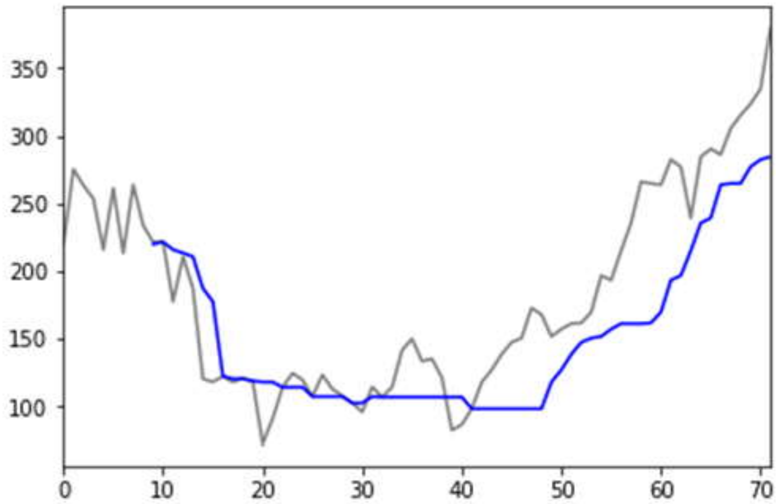
Output:



We can also use quantile on the data

```
data['Price'].plot(color = 'gray')
rolling_window_10.quantile(.25)['Price'].plot(color = 'blue')
```

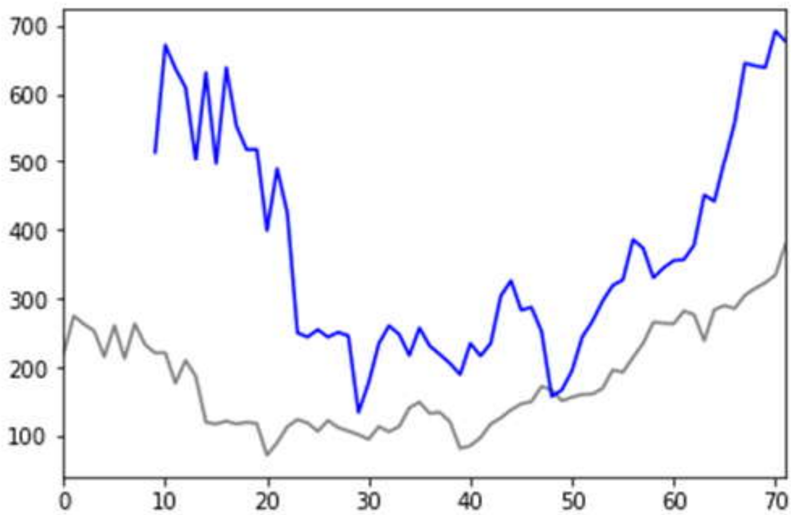
Output:



We can even give our own function

```
data['Price'].plot(color = 'gray')
rolling_window_10['Price'].apply(lambda    x    :
x[0]*math.log10(x[0])).plot(color = 'blue')
```

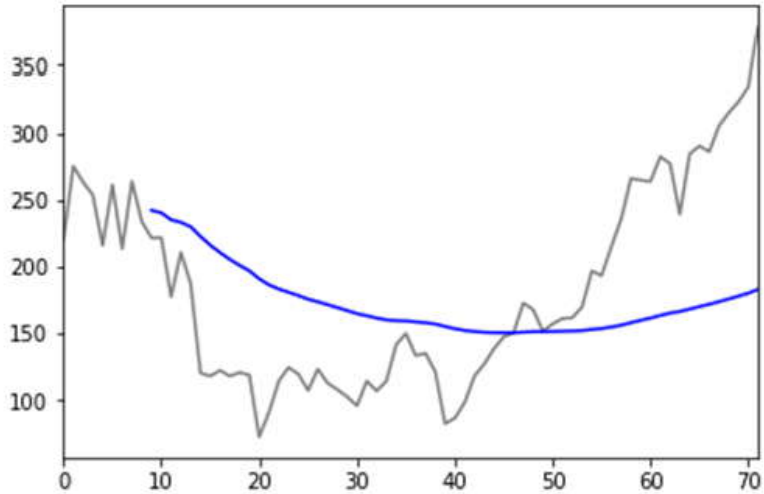
Output:



Now, let's take some example of expending window

```
expending_window = data.expanding(min_periods = 10).mean()
data['Price'].plot(color = 'gray')
expending_window['Price'].plot(color='blue')
```

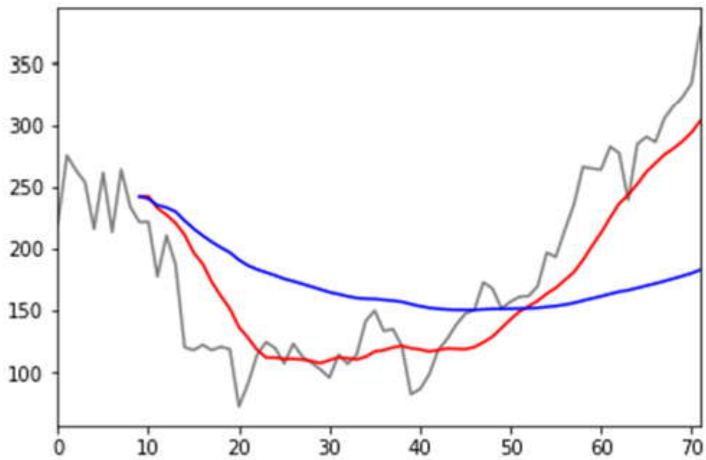
Above code will create an expending window which represents cumulative effect on the data.



We can observer both rolling and expending window together.

```
data['Price'].plot(color = 'gray')
rolling_window_10.mean()['Price'].plot(color = 'red')
expending_window['Price'].plot(color='blue')
```

Output:



12.3 Correlation

Time series data can have a lot of internal structure. For example, sales of winter clothes are higher in some month.

We use autocorrelation function to find the structure in timeseries data. For shorter time we can use partial correlation.

Let's import library first

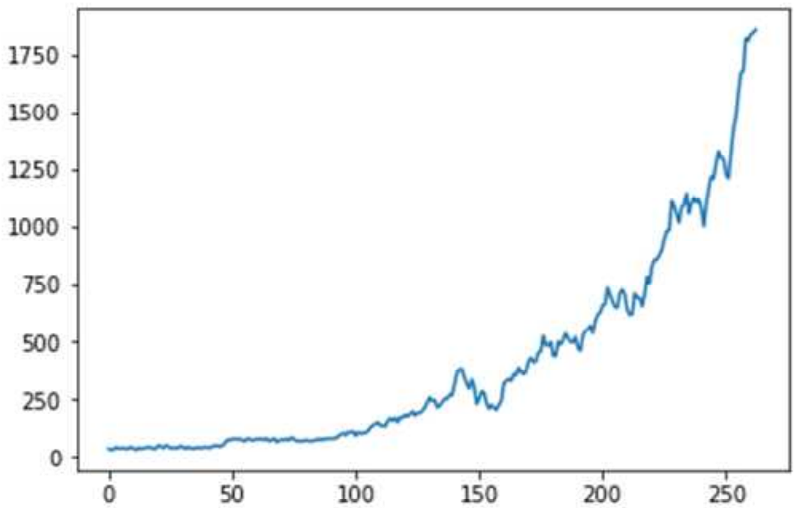
```
from statsmodels.tsa import stattools
```

Read the stock data from a file

```
stock_data= pandas.read_csv('./data/stock_data_2.csv')
```

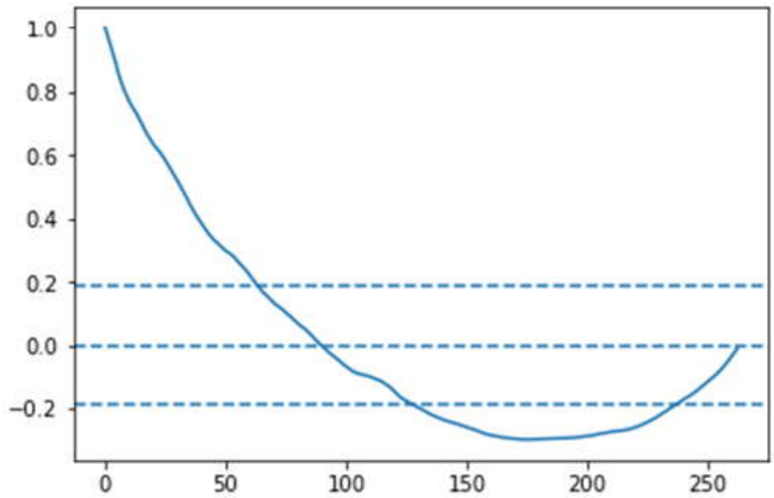
First plot the data by using plot() function

```
plt.plot(stock_data['Price'])
```



This is a data from stock market having 243 observations. Let's apply correlation and check the result.

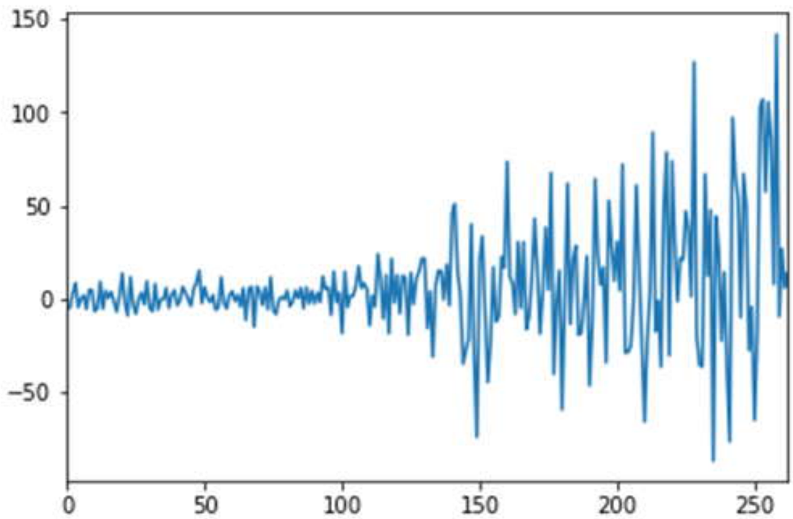
```
autocorr_result = stattools.acf(stock_data['Price'], nlags
= 5000)
plt.plot(autocorr_result)
plt.axhline(y=0,linestyle='--')
plt.axhline(y=-3/
numpy.sqrt(len(stock_data['Price'])),linestyle='--')
plt.axhline(y=3/
numpy.sqrt(len(stock_data['Price'])),linestyle='--')
```



Initially, the correlation was high but it goes decreasing for higher value. We can see from the above that initially stock prices are correlated. But as the stock grows there is less correlation between the values. For some we can see negative correlation.

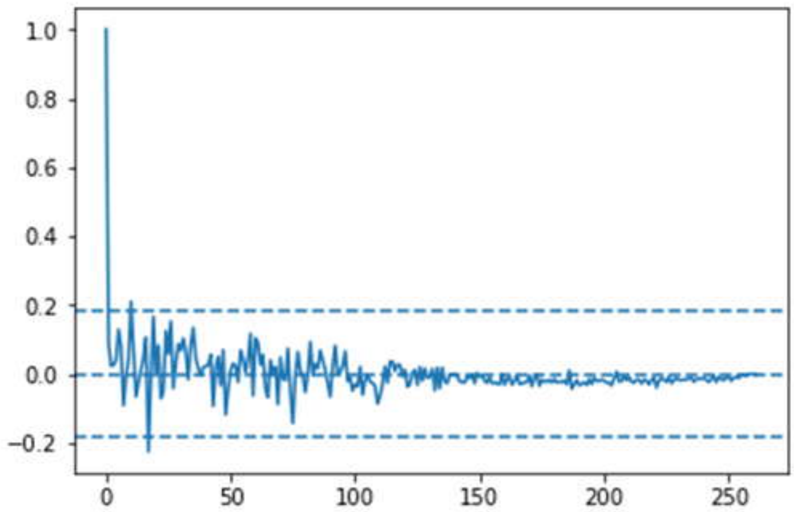
Take the difference of immediate value and create new data.

```
diff_stock_Data = stock_data['Price'] -
stock_data['Price'].shift()
diff_stock_Data.plot()
diff_stock_Data.dropna(inplace = True)
```



Now, again check the auto correlation on this data

```
autocorr_result = stattools.acf(diff_stock_Data, nlags = 5000)
plt.plot(autocorr_result)
plt.axhline(y=0,linestyle='--')
plt.axhline(y=-3/
numpy.sqrt(len(diff_stock_Data)),linestyle='--')
plt.axhline(y=3/numpy.sqrt(len(diff_stock_Data)),linestyle='-
-')
```



12.4 Time Series Forecasting

Time series forecasting is a way to predict the value of a variable in future. To apply any models of time series forecasting we need to make it stationary. Stationary means time series should have constant mean, constant variance and constant autocorrelation. We need to remove seasonality and trends from the data. Seasonality can be additive or multiplicative. We use different transformation on data to remove trends and seasonality. We can test the time series stationary or not by using Dickey-Fuller test.

There are multiple methods and tricks present to make the time series stationary. We need to remove seasonal elements and trends before making any forecast. Most of the real-world time series data contains both seasonality and trends. Time series forecasting model does not need these properties for better forecasting. To detrend the time series we can take moving average. This can be done by window functions described in previous section of this chapter. We can also use linear regression and fit a line along with the data. Removing seasonality is difficult and it needs a lot of domain knowledge. For example, in a drug sale, seasonality may exist due to winter in each year. We can use locally weighted scatterplot smoothing in some cases to remove seasonality.

After making time series stationary we can move to ARIMA model. It contains two types of processes. First is Moving Average(MA). This process is defined by using the equation:

$$X_T = \mu + \theta_0\varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \ldots + \theta_k\varepsilon_{t-k}$$

μ is the mean and θ is the parameter.

So, value X_T is equal to mean value plus error terms in the past data.

If we are using data which is centered at zero or mean is zero,

$$X_T = \theta_0\varepsilon_t + \theta_1\varepsilon_{t-1}$$

Another process in ARIMA is **Auto Regressive (AR)** process. Autoregressive process is represented by:

$$X_T = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \theta_k X_{t-k} + \varepsilon_k$$

So, value at time t depends on previous time. In autoregressive process past value affect the current value. Error term is also present.

ARIMA stands for

AR = Auto Regressive term

I = Differencing (due to de trending)

MA = Moving Average term

ARIMA model is useful when time series is stationary. If time series is not stationary, ARIMA model won't be much useful. Based on the data, we can decide which variant or setting in ARIMA model to use. First, we need to check the trends in data and do appropriate transformations. Some library uses maximum likelihood estimation to get correct parameters for ARIMA model.

Let's build our first ARIMA model. We will take a data and do different transformation on it. Finally, we will apply ARIMA model on it to make time series forecasting.

Data is an Air Passenger data. It contains monthly passengers of an airline from Jan-49 to Dec-60. Data is available publicly available on https://datamarket.com.

```
air_passengers =
pandas.read_csv('.\\data\\air_passengers.csv',parse_dates=['Month'],
index_col=0)
```

We have used Month as date type and used this column as index for this dataframe.

Check the initial records in the data:

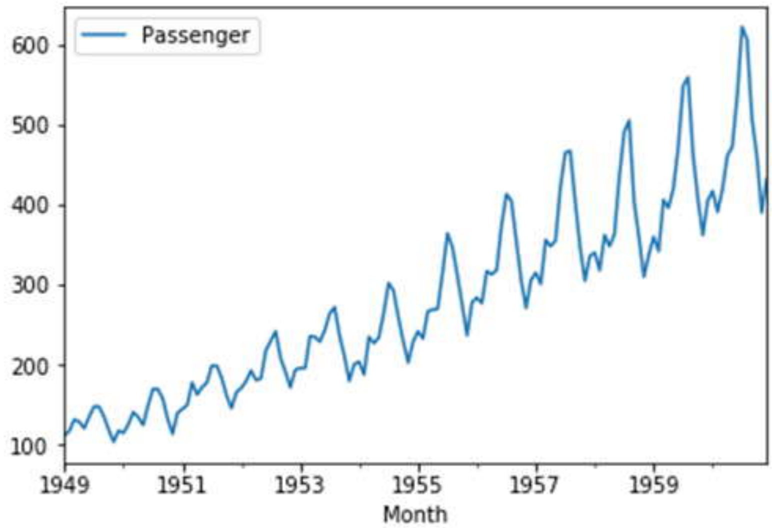
```
air_passengers.head()
```

Now, our task is to make this time series stationary. First, we check that whether the current time series is stationary or not. A time series is stationary if the statistical properties remains constant over the time. Time series should have the following properties:

1. Constant mean
2. Constant variance
3. Autocovariance independent on time

To do the analysis on our data, first plot it

```
air_passengers.plot()
```



We can see that values are increasing over time. There exist seasonal trends, but overall value is increasing. We can use Dickey-Fuller Test to check time series is stationary or not. In this test we take null hypothesis as time series is not stationary. Alternative hypothesis would be time series is stationary. Results contain test statistics and critical values for confidence levels. If the critical value is greater than the test statistics, we can reject the null hypothesis and accepts the alternative one.

Import library for Dickey-Fuller Test

```
from statsmodels.tsa.stattools import adfuller
```

Function to get and print test result

```
def test_timeseries_stationary(ts):
    ts = ts['Passenger']
    print('Dickey-Fuller Test:')
    dickey_fuller_test = adfuller(ts, autolag='AIC')
    dickey_fuller_output =
pandas.Series(dickey_fuller_test[0:4], index=['Test
Statistic','p-value','Number of Lags Used','Observations
Used'])
```

```
    for key,value in dickey_fuller_test[4].items():
    dickey_fuller_output['Critical Value (%s)'%key] = value
    print(dickey_fuller_output)
```

Now, apply this test on existing data

```
test_timeseries_stationary(air_passengers)
```

We will get these results:

```
Dickey-Fuller Test:
Test Statistic            0.815369
p-value                   0.991880
Number of Lags Used      13.000000
Observations Used       130.000000
Critical Value (1%)      -3.481682
Critical Value (5%)      -2.884042
Critical Value (10%)     -2.578770
```

As we can see Test Statistics is higher than critical values. So, this given raw time series is not stationary.

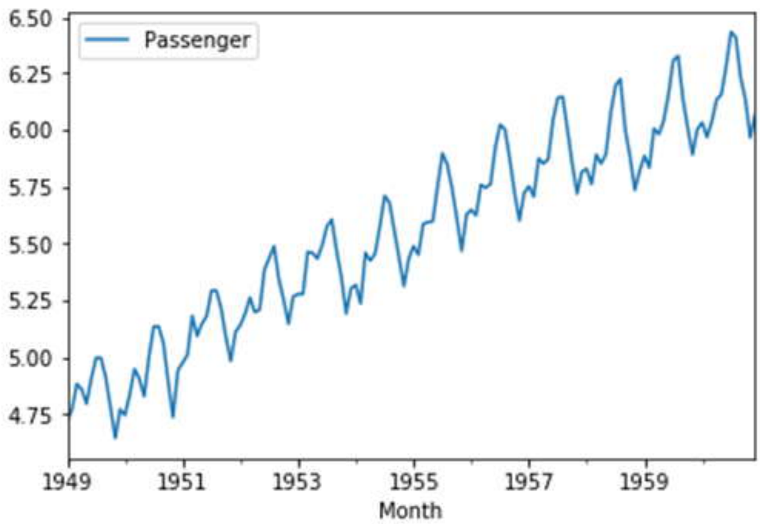
Now, we will do some transformation and again check the result of this test.

First take the log of the existing data

```
air_passengers_log_value = numpy.log(air_passengers)
```

Generate plot of new data

```
air_passengers_log_value.plot()
```

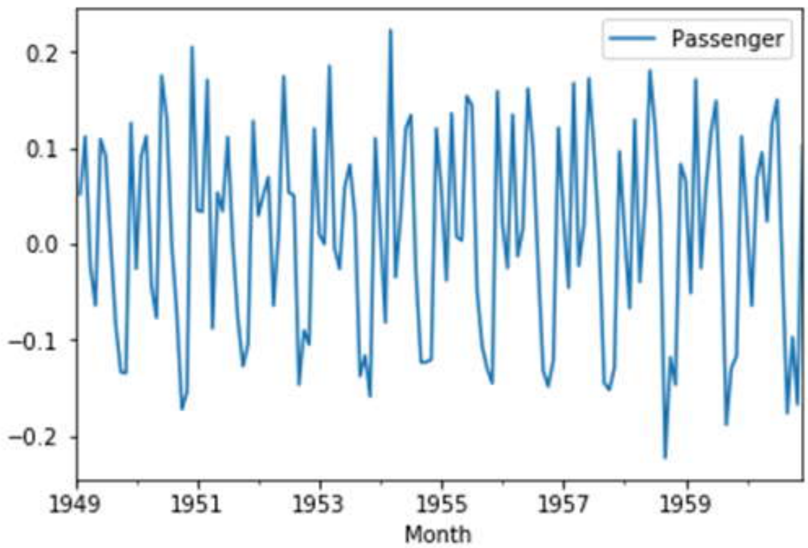


Now, get a new series with a difference of consecutive values.

```
air_passengers_log_value_diff = air_passengers_log_value -
air_passengers_log_value.shift()
```

Plot this data

```
air_passengers_log_value_diff.plot()
```



This data contains NaN value at first record. Because the difference is not defined for that. We can remove that value and apply Dickey-Fuller Test.

```
air_passengers_log_value_diff.dropna(inplace=True)
test_timeseries_stationary(air_passengers_log_value_diff)
```

Result for above test

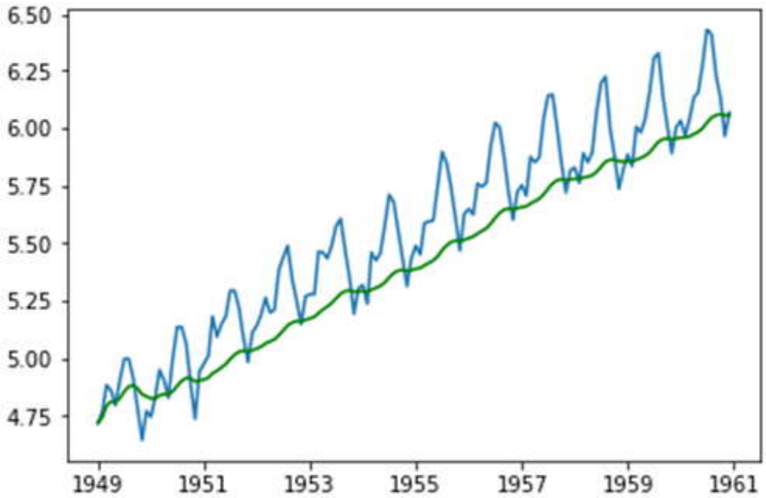
```
Dickey-Fuller Test:
Test Statistic              -2.717131
p-value                      0.071121
Number of Lags Used         14.000000
Observations Used          128.000000
Critical Value (1%)         -3.482501
Critical Value (5%)         -2.884398
Critical Value (10%)        -2.578960
```

These are somewhat desired results. Here, test statistics is smaller than 5% critical value. So, we can say with 95% of confidence that this time series is stationary.

Let's do some more transformation on this data. We can apply aggregation,

smoothing and regression fitting to make this data stationary. Let's apply moving average techniques. At each point take average of past 12 values. This will represent yearly average at that point. We can use a modified technique for moving average. New values are more important than the old one. So, we can take weighted moving average instead.

```
exponential_weighted_average =
pandas.ewma(air_passengers_log_value, halflife=12)
plt.plot(air_passengers_log_value)
plt.plot(exponential_weighted_average, color='Green')
```



Create a difference series

```
exponential_weighted_average_diff = air_passengers_log_value
- exponential_weighted_average
```

Now, check the result of test

```
test_timeseries_stationary(exponential_weighted_average_diff)
```

```
Dickey-Fuller Test:
Test Statistic -3.601262
p-value 0.005737
Number of Lags Used 13.000000
Observations Used 130.000000
Critical Value (1%) -3.481682
Critical Value (5%) -2.884042
Critical Value (10%) -2.578770
```

From above result we can see that test statistics is less than 1% critical value. So, we can say with 99% confidence that our time series is stationary.

Now, let's apply ARIMA model for time series forecasting. ARIMA model contains three parameters.

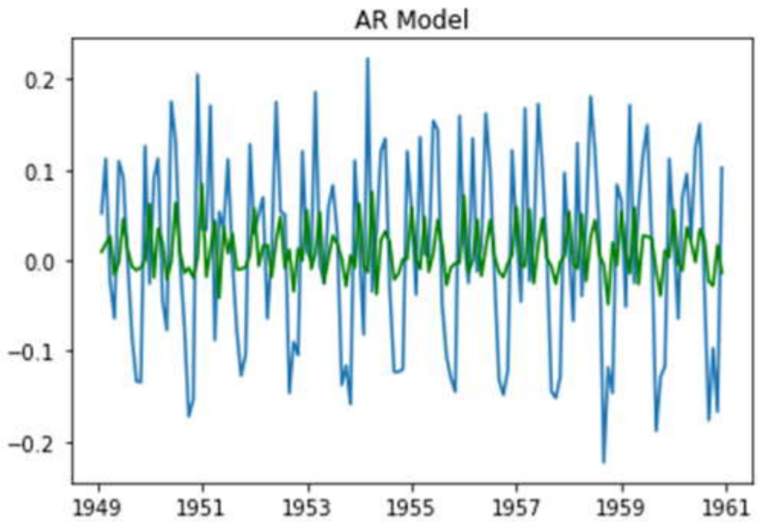
1. **AR (Number of Auto regressive term):** These are the lag of dependent variable.
2. **I (Number of difference):** Number of non-seasonal differences.
3. **MA (Moving average):** lagged errors.

Now, to make ARIMA model first import the library needed

```
from statsmodels.tsa.arima_model import ARIMA
```

First, make model by using AR only. We will put 0 on MA parameter.

```
#Auto Regressive Model
ARIMA_model_1 = ARIMA(air_passengers_log_value, order=(2,
1, 0))
results_AR_1 = ARIMA_model_1.fit(disp=-1)
plt.plot(air_passengers_log_value_diff)
plt.plot(results_AR_1.fittedvalues, color='Green')
plt.title('AR Model')
```



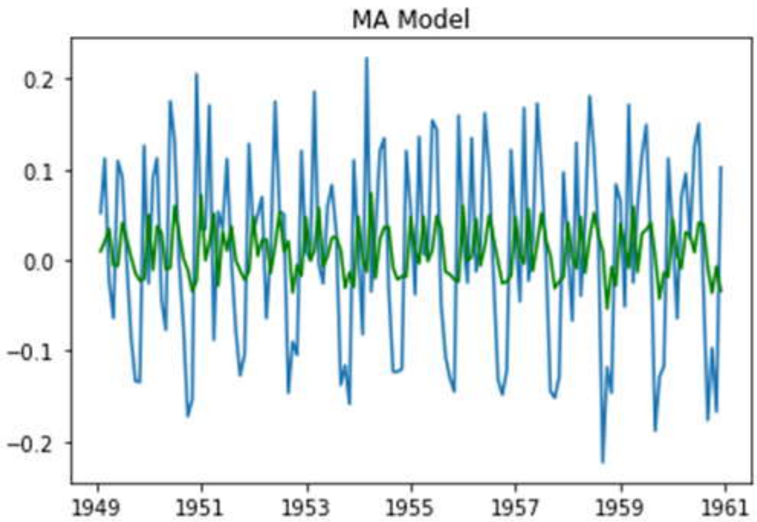
Calculate RSS for this model:

```
print('Residual Sum Of Square: %.6f'% sum((results_AR_1.
fittedvalues-air_passengers_log_value_diff['Passenger'])**2))
```

Residual Sum Of Square: 1.502303

Now, create model by using Moving average only:

```
#Moving Average Model
ARIMA_model_2 = ARIMA(air_passengers_log_value, order=(0,
1, 1))
results_MA_2 = ARIMA_model_2.fit(disp=-1)
plt.plot(air_passengers_log_value_diff)
plt.plot(results_MA_2.fittedvalues, color='Green')
plt.title('MA Model')
```



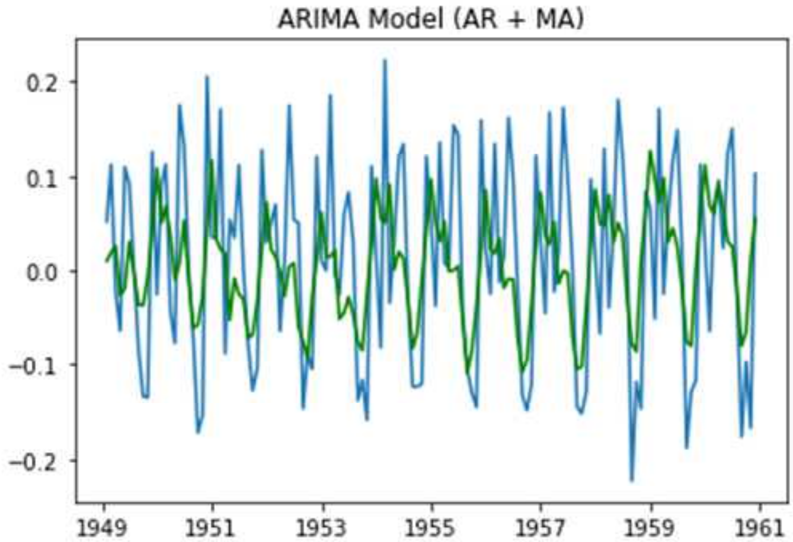
Check the error

```
print('Residual Sum Of Square: %.6f'% sum((results_MA_2.
fittedvalues-air_passengers_log_value_diff['Passenger'])**2))
```

Residual Sum Of Square: 1.524538

There is no much difference is Error (by checking RSS value). Now, create ARIMA model considering both AR and MA.

```
#ARIMA Model (AR + MA)
ARIMA_model_3 = ARIMA(air_passengers_log_value, order=(2,
1, 1))
results_ARIMA_3 = ARIMA_model_3.fit(disp=-1)
plt.plot(air_passengers_log_value_diff)
plt.plot(results_ARIMA_3.fittedvalues, color='Green')
plt.title('ARIMA Model (AR + MA)')
```



Check error:

```
print('Residual Sum Of Square: %.6f'% sum((results_MA_2.
fittedvalues-air_passengers_log_value_diff['Passenger'])**2))
```

Residual Sum Of Square: 1.459731

Compare the RSS of above three different models:

Model	Residual Sum of Squares
AR	1.502303
MA	1.524538
AR + MA	1.459731

So, we can see from above result that third model (AR + MA) is best for our example.

Now, we have result from this model. Let's take it to original form to get the correct output predictions.

First, create a new series of values generated from the model:

```
ARIMA_diff_values =
pandas.Series(results_ARIMA_3.fittedvalues, copy=True)
```

To convert differencing to log scale, first we will add differences to the base numbers. This can be done by cumsum() function in pandas.

```
ARIMA_diff_values_cumsum = ARIMA_diff_values.cumsum()
```

Now, we have to add these values to our first base number

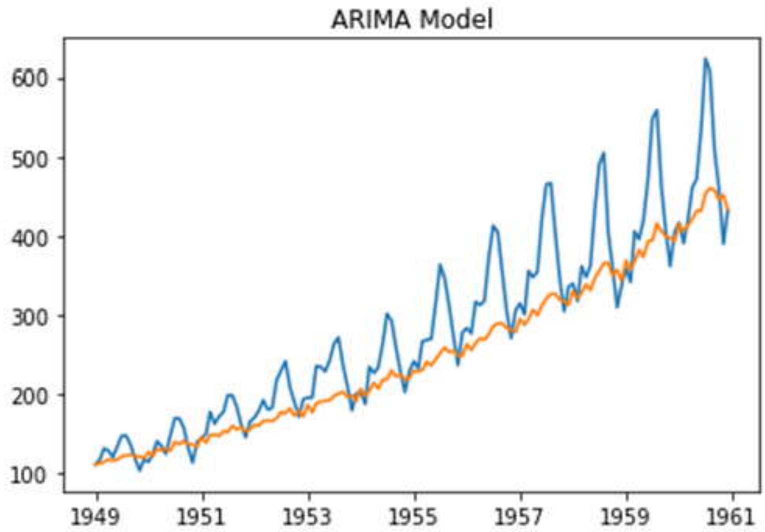
```
ARIMA_log =
pandas.Series(float(air_passengers_log_value.ix[0]),
index=air_passengers_log_value.index)
ARIMA_log =
ARIMA_log.add(ARIMA_diff_values_cumsum,fill_value=0)
```

Till now we are working on log. Now, take the exponent of the values.

```
ARIMA_Result = numpy.exp(ARIMA_log)
```

Plot the result

```
plt.plot(air_passengers)
plt.plot(ARIMA_Result)
plt.title('ARIMA Model')
```



Check for overall error

```
print('Root Mean Squared Error: %.6f'% numpy.sqrt
(sum((ARIMA_Result-air_passengers['Passenger'])**2)/
len(air_passengers['Passenger'])))
```

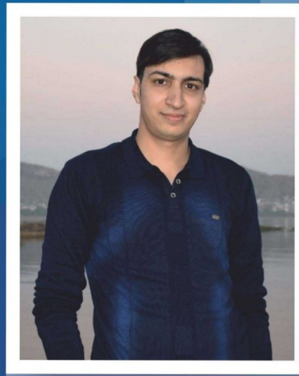
```
Root Mean Squared Error: 52.489088
```

We can create models with different settings and compare RMSE.

CD INCLUDED

MACHINE LEARNING WITH PYTHON

An Approach to Applied Machine Learning



Abhishek Vijayvargia is a Data Scientist. He worked in IT industry and helped in solving real time problems related to Data Science and Machine Learning. He worked on analytics problems related to transportation, government process, manufacturing, oil and gas, IoT, pharmaceuticals, shipping. He has completed his Masters from Indian Institute of Technology(IIT), Kanpur in Artificial Intelligence. His research interests are Distributed Machine Learning, Deep Learning, Stream Processing and Blockchain. He worked as mentor for various Machine Learning projects and trained others in Algorithm, Competitive Programming and Data Science.

This book provides concept of machine learning with mathematical explanation and programming examples. Every chapter starts with fundamentals of the technique and working example on real world dataset. Along with the advice on applying algorithms, each technique is provided with advantages and disadvantages on the data.

In this book we provide code examples in Python. Python is the most suitable and worldwide accepted language for this. First, it is free and open source. It contains very good support from open community. It contains a lot of library, so you don't need to code everything. Also, it is scalable for large amount of data and suitable for big data technologies.

Salient feature

- Covers all major areas in Machine Learning.
- Topics are discussed with graphical explanations.
- Comparison of different Machine Learning methods to solve any problem.
- Methods to handle real world noisy data before applying any Machine Learning algorithm.
- Python code example for each concept discussed.
- Jupyter notebook scripts are provided with dataset used to test and try the algorithms.

www.bpbonline.com

₹ 297/-

BPB PUBLICATIONS
20 Ansari Road, Darya Ganj, New Delhi-110002
sales@bpbonline.com

ISBN 978-93-8655-193-1